2000-03-01

QOS management with adaptive routing for next generation internet

Quek, Henry C.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

QOS MANAGEMENT WITH
ADAPTIVE ROUTING
FOR
NEXT GENERATION INTERNET

by

Henry C. Quek

March 2000

Thesis Advisor: Geoffrey Xie
Second Reader: Bret Michael

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE QOS MANAGEMENT WITH ADAPTIVE ROUTING FOR NEXT GENERATION INTERNET	5. FUNDING NUMBERS
6. AUTHOR(S) Henry C. Quek	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA and NASA	10. SPONSORING/MONITORING AGENCY REPORT NUMBER G417

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE Statement A

13. ABSTRACT *(maximum 200 words)*

Up till today, the Internet only provides best-effort service, where traffic is processed as quickly as possible, with no guarantee as to timeliness or actual delivery. As the Internet develops into a global commercial infrastructure, demands for guaranteed and differentiated network quality of service (QoS) will increase rapidly. Several QoS service models have been developed to provide and support QoS in the Internet, namely: Integrated Service (IntServ), Differentiated Service (DiffServ) and MultiProtocol Label Switching (MPLS). QoS routing, such as Widest-Shortest Path, Shortest-Widest Path and Shortest-Distance Path, is required in order to support QoS and optimize the network resource utilization.

The Server and Agent based Active network Management (SAAM) system is a network management system designed for the next generation Internet. It is capable of supporting all types of service. It will be able to control and optimize the utilization of the network through resource allocation and adaptive QoS routing.

This thesis describes a design and implementation of the QoS Management component of a SAAM Server. This component optimizes the utilization of network resources and supports the various service classes in a cohesive manner. It utilizes an adaptive routing strategy to balance the network load.

14. SUBJECT TERMS Next Generation Internet, Integrated Services, Differentiated Service, MPLS, Quality of Service, Flows, Networks, Routing	15. NUMBER OF PAGES 287
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

QOS MANAGEMENT WITH ADAPTIVE ROUTING FOR NEXT GENERATION INTERNET

Henry C. Quek
Republic of Singapore's Ministry of Defense
B. Eng. University of Leeds (UK), 1995

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

ABSTRACT

Up till today, the Internet only provides best-effort service, where traffic is processed as quickly as possible, with no guarantee as to timeliness or actual delivery. As the Internet develops into a global commercial infrastructure, demands for guaranteed and differentiated network quality of service (QoS) are increasing rapidly. Several QoS service models have been developed to provide and support QoS in the Internet, namely: Integrated Service (IntServ), Differentiated Service (DiffServ) and MultiProtocol Label Switching (MPLS). QoS routing, such as Widest-Shortest Path, Shortest-Widest Path and Shortest-Distance Path, is required in order to support QoS and optimize the resource utilization.

The Server and Agent based Active network Management (SAAM) system is a network management system designed for the next generation Internet. It is capable of supporting all types of service. It will be able to control and optimize the utilization of the network through resource allocation and adaptive QoS routing.

This thesis describes a design and implementation of the QoS Management component of a SAAM Server. This component optimizes the utilization of network resources and supports the various service classes in a cohesive manner. It utilizes an adaptive routing strategy to balance the network load.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge the financial support of the Defense Advanced Research Projects Agency and the National Aeronautics and Space Agency for the purchase of the equipment used in this thesis.

Many thanks to Dr. Geoffrey Xie for his constant willingness to teach and advise me with my research. His guidance and enthusiasm has helped to carry this project to completion.

I would also like to acknowledge the contribution and help that Mr. Cary Colwell and Mr. Liu Wenzhi had rendered in various parts of my project.

I thank the Ministry of Defense of Singapore for this opportunity that they have given me to learn and further my education in a conducive environment like this - The Naval Postgraduate School.

I am most grateful to my wife and daughter, Celestin and Vanessa, whose love and support were essential during the development and writing of this thesis. Their patience throughout my course of study is truly appreciated.

I would also like to thank my parents for their love and sacrifice, without whom neither of us would have the opportunity that we are both enjoying today.

Last but not the least, I thank God, for his grace, without which I would be lost and would not have this opportunity to do what I have accomplished.

I. INTRODUCTION

A. BACKGROUND

The Internet started in the mid 1980s with ARPANET and NSFNET interconnected together. Many other networks joined in later. Up till today, the Internet only provides best-effort service, where traffic is processed as quickly as possible, with no guarantee as to timeliness or actual delivery. As the Internet developed into a global commercial infrastructure, demands for guaranteed and differentiated quality of service (QoS) have increased rapidly. It is now apparent that the control system of the Internet needs to be upgraded to provide the various types of QoS demands.

Several QoS service models have been developed recently to define the scope and guide the implementation of QoS in the Internet. They are: Integrated Service (IntServ), Differentiated Service (DiffServ), and MultiProtocol Label Switching (MPLS). IntServ is characterized by resource reservation while the later two are characterized by per hop behaviors [6].

QoS routing, such as Widest-Shortest Path (WSP), Shortest-Widest Path (SWP) and Shortest-Distance Path (SDP), is required in order to support QoS and optimize the utilization of network resources such as link bandwidth. WSP selects a path with the least number of hops, while SWP selects a path with the largest available bandwidth. Shortest-Distance Path selects a path with the lowest distance computed using a predefined distance function.

Server and Agent based Active network Management or SAAM, is a network management system for the next generation Internet, which will be able to support all service classes that are defined for the future Internet. It will be able to control and optimize the utilization of the network through resource allocation and adaptive QoS routing.

B. AN OVERVIEW OF SAAM

SAAM is an intelligent network management system, which comprises a hierarchy of servers and lightweight routers that are partitioned into regions (see Figure

1.1). The key feature of SAAM is that all network management decisions are carried out at the servers, which lightens the workload on the routers (hence we called them lightweight routers). SAAM also allows sophisticated software solutions to be deployed to servers to implement network QoS (both IntServ and DiffServ) and to optimize the use of resources without overextending the routers.

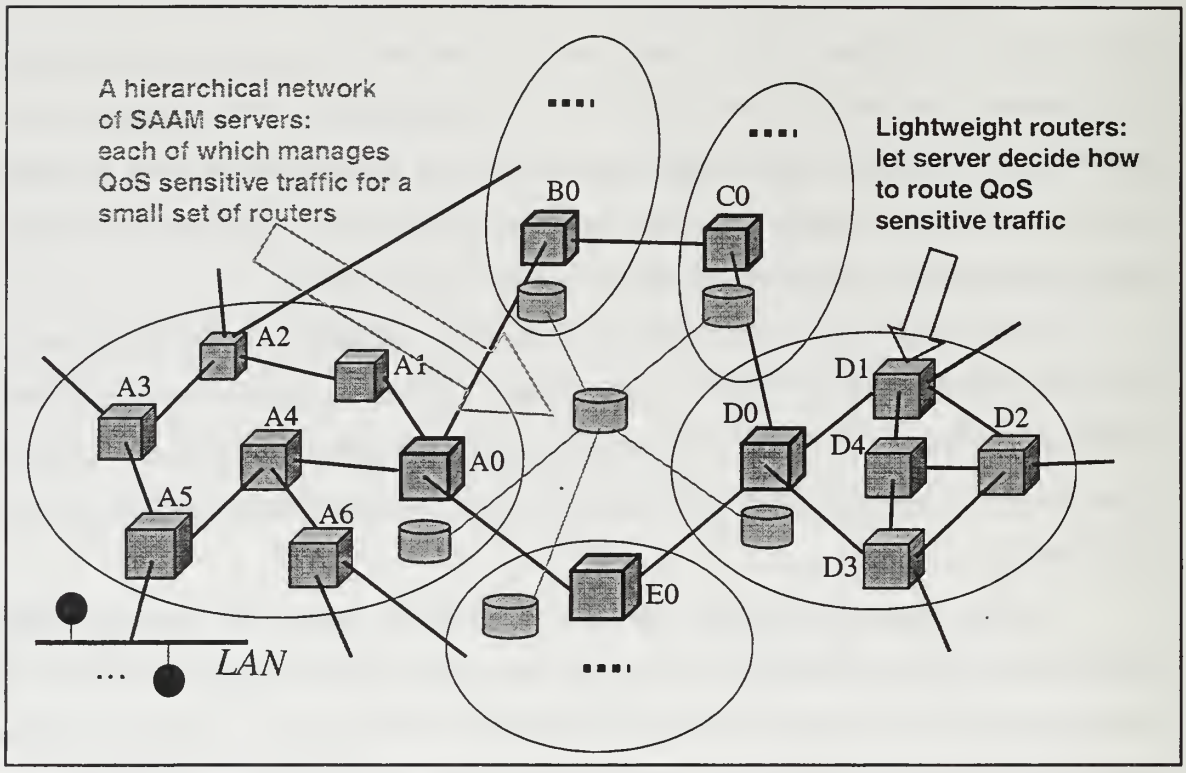


Figure 1.1 – An example of SAAM network architecture

1. SAAM Server

Each SAAM Server is analogous to a helicopter that monitors and directs the commuting traffic over an area. It maintains a global view about the performance of the routers' data paths in its region using a path information base (PIB). With this view, the server will be able to carry out QoS routing or re-routing, and direct traffic to the best path between a source and destination. The best path may be the least congested path among all possible paths such that the transit time meets the application requirement, or one that will optimize the resources availability.

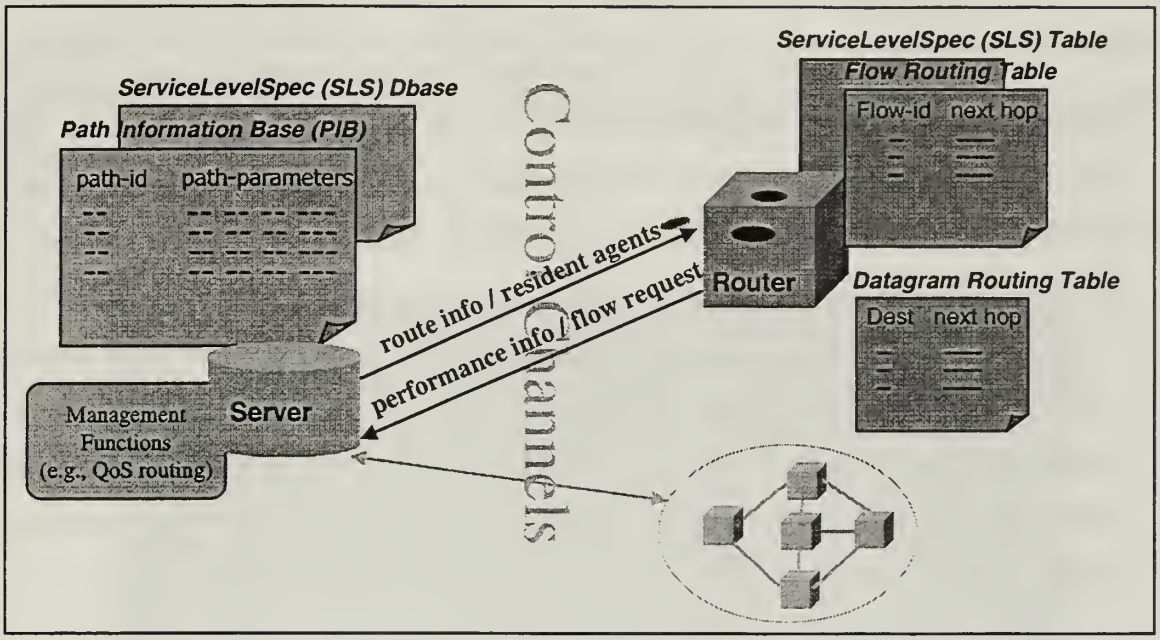


Figure 1.2 - SAAM Server and Router interaction

2. SAAM Router

Each SAAM Router is analogues to a local traffic controller at an intersection or access ramp. All traffic that wishes to enter the Internet will have to be admitted by the routers, which in turn seek approval from the server that is in control of the region (see Figure 1.2). If an IntServ flow is admitted, a path along with a flow id will be selected for it by the server. If necessary, messages will be sent from the server to all the routers along the path to update their Flow Routing Tables. If a new DiffServ flow is admitted, a Service Level Spec (*SLS)* will be assigned to it by the server. A message will be sent from the server to the edge router to update their SLS Tables.

C. GOAL OF THE SAAM PROJECT

The goal of the SAAM project, under which this research is conducted, is to provide various types of QoS to applications while optimizing network resource utilization through a central resource management system with adaptive routing. Specifically, SAAM seeks to achieve the following objectives:

1. Integrated And Differentiated Services

In SAAM, each link is logically partitioned into various service level pipes, with a specific share of network resources allocated to each, to support various service classes. A SAAM server supports an IntServ flow by finding a feasible path that is able to support the QoS requirements of the flow and making the necessary reservation of resources along the path. DiffServ flows are more difficult to manage because there are two types of DiffServ flows: *Static SLS* and *Dynamic SLS*. For a Static SLS flow, the admission control and policing are actually done at the edge router where the flow enters the network. However, for a dynamic SLS flow, the server will carry out the admission control and delegate the policing to the edge router. SAAM Servers will maintain and update the SLS Table needed for both static and dynamic SLS at the routers.

2. Optimal Use Of Resources

By maintaining an accurate region-wide view of the network performance and resource availability, SAAM server will be able to dynamically route or re-route traffic to optimize the use of its resources. This is the major advantage that SAAM has over current network architecture, which is based on stand-alone routers.

3. Automated Fault Detection And Timely Recovery

As mentioned earlier, SAAM Server is the decision-making element that manages the whole network region. It controls all the routers in its region and maintains the path information base for all the routes. Hence, it is critical that any fault in the region, particularly in the server, be detected timely and recovery action taken as soon as possible. The fault detection method employed in SAAM is based on an Accelerated Heartbeat Protocol [17].

4. Support of Incremental Deployment

The SAAM architecture is designed to allow network engineers to incrementally replace the legacy network infrastructure, providing improvements of network performance to those ISPs that adopt SAAM. An ISP has total control over the operation

of its own SAAM server (see Figure 1.3). The super server acts like an advisory center providing only performance enhancing advice to the internal servers. Incremental deployment of SAAM requires it to support and cooperate with legacy systems in terms of protocols.

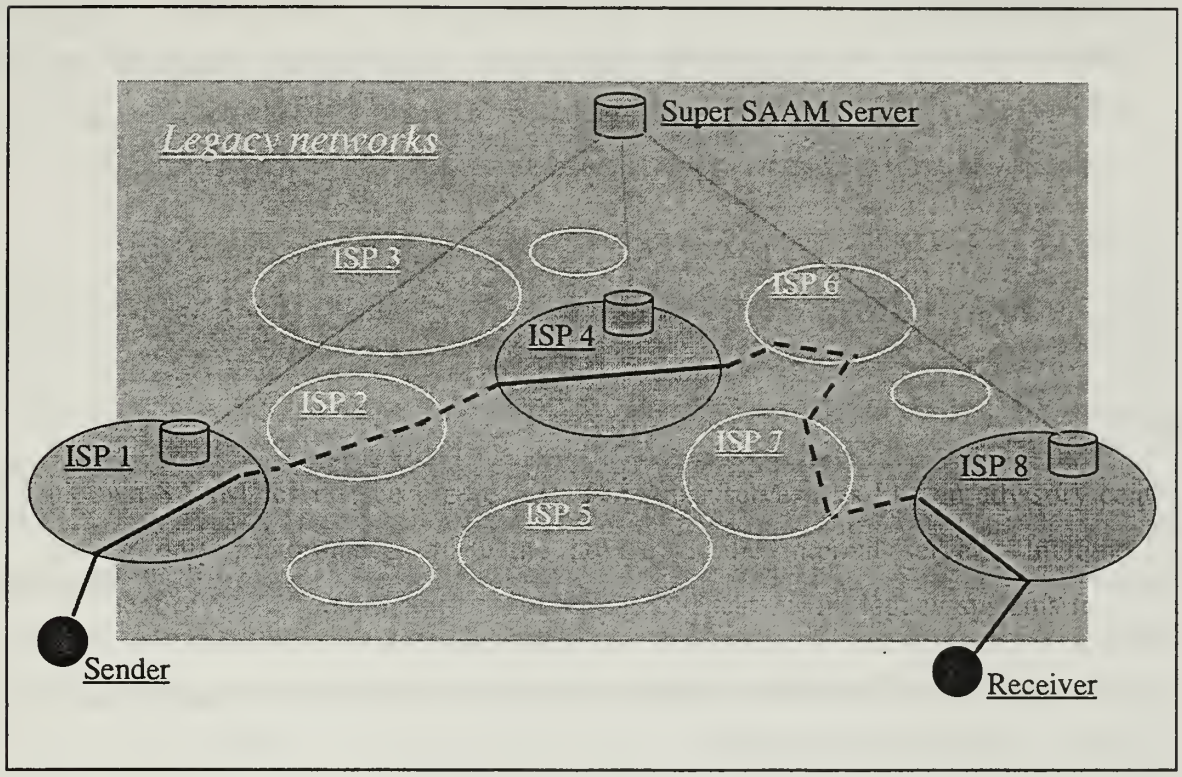


Figure 1.3 – Incremental Deployment of SAAM

D. SCOPE OF THIS THESIS

The primary goal of this thesis is to develop an efficient QoS management algorithm and integrate it into the existing SAAM architectural design. In order to manage the resources efficiently, a SAAM Server will need to be able to adapt its routing algorithm under varying network conditions. Hence this thesis also studies the use of an adaptive QoS routing strategy to optimize the network utilization.

E. MAJOR CONTRIBUTIONS OF THIS THESIS

The results of this research have the potential to be integrated into a system that will benefit every user of the Internet. Though the concepts of DiffServ and IntServ have been developed by the Internet Engineering Task Force (IETF), the actual implementation of them has not been widespread. The feasibility of integrated support of DiffServ and IntServ has been investigated for a single SAAM network region. The results provide strong evidences that the two service models can coexist in SAAM.

F. ORGANIZATION

This thesis is divided into several chapters.

- Chapter II discusses the related topics of QoS routing and QoS models.
- Chapter III describes the design of a QoS Management component for SAAM.
- Chapter IV describes the implementation of the QoS Management component.
- Chapter V discusses the tests conducted.
- Chapter VI concludes with words on the results obtained, lessons learned and the future work needed.

II. RELATED TOPICS

In order for SAAM to support and integrate well with the Internet, SAAM developers need to have a basic knowledge of the current Internet routing protocol (see appendix A) and its future developments. In this chapter, various QoS routing algorithms and the three most accepted Internet QoS models are discussed.

A. QUALITY OF SERVICE ROUTING

Network usage has grown rapidly over the years and demands are ever increasing. Despite the advances of technology in expanding the physical bandwidth limitation of the network, there is always a tendency for it to be overloaded. Therefore applications requiring certain network performance guarantees will have unsatisfactory results using best effort networks. The solution to this problem, is Quality-of-Service (QoS).

Routing deployed in the current Internet is focused mainly on connectivity and typically supports only the "best effort" datagram service (see Appendix A). The routing protocol uses "shortest path routing", which chooses an optimized path based on a single arbitrary metric (e.g. administrative weight or hop count). QoS or QoS-based Routing, as defined in RFC2386, is a routing mechanism under which paths for flows are determined based on some knowledge of resource availability in the network as well as the QoS requirement of flows.

Many have thought that Resource ReSerVation Protocol (RSVP) [8] is a form of QoS routing. The fact is that RSVP is just a protocol that supports the reservations of resources across an IP network. RSVP provides a method for the application to interact with the network for requesting and reserving network resources; it does not provide a mechanism for determining a network path that has adequate resources to accommodate the requested QoS.

QoS routing is different from RSVP. It allows the determination of a path that has a good chance of accommodating the requested QoS. However, it does not include a mechanism to reserve the required resources.

1. Goals of QoS Routing:

The goals of QoS routing are:

- To find a feasible path; A path is feasible if the unused bandwidth of all links on the path is higher than the requested bandwidth [3].
- To select a feasible path (when more than one exists) that will lead to a better overall resource efficiency and optimize resource utilization.

According to RFC 2386, QoS routing must extend the current Internet routing paradigm in three basic ways:

- It must be able to support traffic using integrated-service class of services, i.e. the integration of QoS services and best effort services. Additional routing metrics, such as transit delay and available bandwidth, may have to be made available and distributed.
- It should maintain the use of current path even when a better path is found so long as it meets the requirements of the existing traffic. This is to avoid unnecessary traffic shifts between alternate paths so as to prevent routing oscillations and prevent variation of delay and jittering which may be experienced by the end user.
- It should support alternate routing which the current Internet protocol does not by keeping a list of possible alternate paths for re-routing.

2. Strategies of QoS Routing

There are three main strategies of QoS routing: Source Routing, Distributed Routing, and Hierarchical Routing. These are classified according to how the state information is maintained and how the search of feasible paths is carried out.

a) Source routing

In Source Routing, each node maintains the complete global state. The global state includes the network topology and state information of every link. A feasible path is computed at the source node based on the global state. A control message is then

sent to establish the path chosen. A link state protocol is used to update the global state in all the nodes along the path.

Source routing avoids the complexity of distributed computing by simply maintaining a global state and computes the path locally. However, the state information at each node has to be current. Failure of this will result in not finding an existing feasible path. Hence, the global state has to be frequently updated, resulting in large overhead and scalability problem.

b) Distributed routing

In Distributed routing, the path is computed by a distributed computation. Control messages are exchanged among the nodes. State information is kept at each node and collectively used for path selection. Routing is done on a hop-by-hop basis.

Because of distributed computing, the response time can be made shorter and the algorithm more scalable. However, when global states at different nodes are inconsistent, loops may occur.

c) Hierarchical routing

In Hierarchical routing, nodes are clustered into hierarchical groups. Each node maintains an aggregated global state, which contains the state information of the nodes in the same group and the aggregate information of other groups. Source routing is used to find a feasible path, which may contain logical nodes representing other groups. A control message is sent along the path to establish it. If the border node of a group representing a logical node receives the message, it expands the path through that group.

Hierarchical routing is used to overcome the problem of scalability that source routing has. It retains many of the advantages of source routing. It has the advantages of distributed routing because many nodes share routing computation.

3. Path Selection Schemes of QoS Routing

There are three main types of QoS routing: Widest-Shortest Path, Shortest-Widest Path, and Shortest-Distance Path. These are classified according to how the state information is maintained and how the search of feasible paths is carried out.

- Widest-Shortest Path (WSP) is one that selects a path with the minimum hop count and, if there are multiple such paths, the one with the largest available bandwidth. This scheme emphasizes preserving network resources by choosing the shortest paths first.
- Shortest-Widest Path (SWP) is one that selects a path with the largest available bandwidth and, if there are multiple such paths, the one with the minimum hop count. This scheme emphasizes load balancing by choosing the widest paths first.
- Shortest-Distance Path (SDP) is one that selects a path with the lowest distance computed such that the distance of a k-hop path P is

$$dist(P) = \sum_{i=1}^{k} \frac{1}{r_i}$$

where r_i is the available bandwidth of link i

This scheme makes a trade-off between SWP and SDP. It favors shortest paths when the network load is heavy and widest paths when the network load is medium.

B. INTERNET QUALITY OF SERVICE MODELS

There are generally three widely accepted QoS service models that are being studied for the Internet: Integrated Service (IntServ), Differentiated Service (DiffServ) and MultiProtocol Label Switching (MPLS). As the future Internet may be comprised of these types of service classes, SAAM developers need to consider how these services are to coexist in one SAAM region. In order to answer this question, one shall first seek to understand what these service models are and how they differ.

1. Integrated Service

Integrated Service or *IntServ* is characterized by resource reservation. The network has to set up paths and reserve resources before application data can be sent. RSVP is the signaling protocol for setting up paths and reserving resources. The philosophy of this model is that routers need to be able to reserve resources in order to

provide special QoS for specific state in the routers. Hence, IntServ capable routers must have flow-specific state in them. IntServ may be viewed as a guaranteed service class which requires fixed delay bound, or a controlled-load service class which requires reliable and enhanced best-effort service.

a) Operation of Integrated Service

IntServ is implemented by four main components: the resource reservation protocol, the admission control, the classifier, and the packet scheduler. Figure 2.1 shows the diagram of an IntServ Router model.

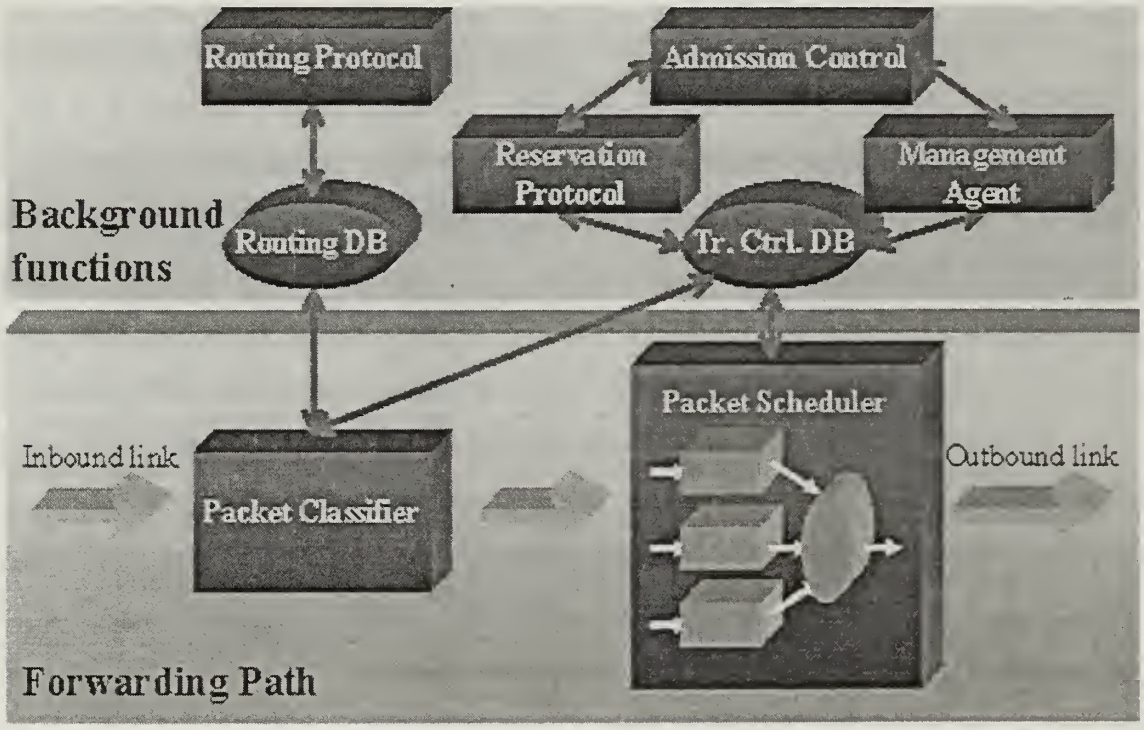


Figure 2.1 - Integrated Service Router Model

A resource reservation protocol is required to set up paths and reserve resources before data can be sent. Admission control decides whether a request for resources can be granted. If granted, the classifier will perform multifield classification and put the packet in a specific queue based on the classification. The packet scheduler will then schedule the packet accordingly to meet the QoS requirements.

b) Problems of Integrated Service

IntServ routes packets on a per flow basis. Hence, the amount of state information increases with the number of flows, and requirements on the routers are high. All routers must have all the four components - RSVP, admission control, MF classification, and packet scheduling - to support IntServ. Therefore, in order for IntServ to work, all routers in the network must be IntServ capable. This means, deployment of IntServ domain has to be done all at the same time. Progressive deployment is difficult though possible.

2. Differentiated Service

Difficulty in implementing and deploying IntServ brought about Differentiated Service or *DiffServ*. In DiffServ, packets are marked differently according to its class specified in the Type of Service (TOS) field within the IP header (see Figure 2.2). This means that each service class of DiffServ has separate queue instead of having one queue per flow for the case of IntServ.

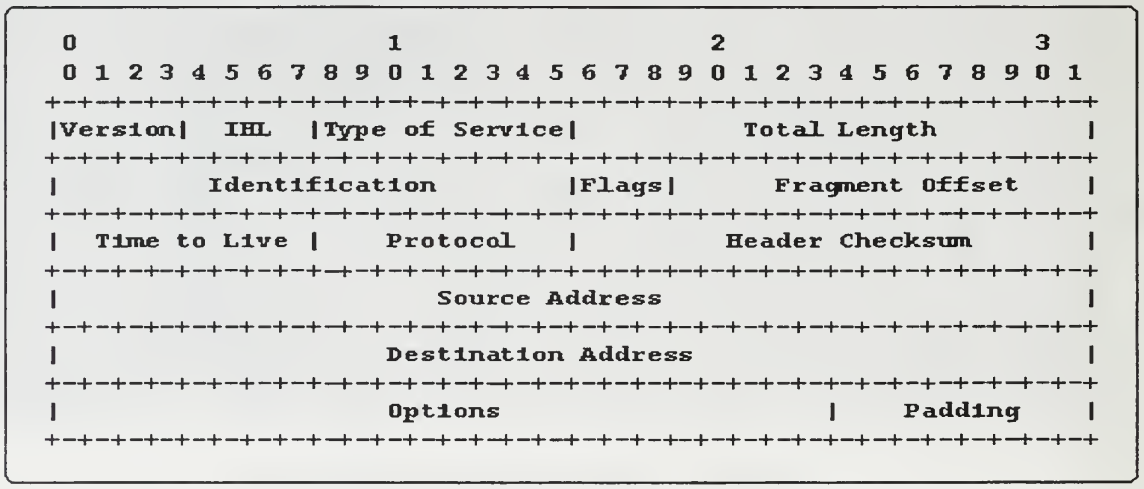


Figure 2.2 - Differentiated Service Field in IPv4 Header

Figure 2.3 shows the specification of the TOS field. PHB field value specifies the Per-hop Behavior (PHB) to be allotted to the packet within the provider's network. Its behavior name are 00000 for default, and 11100 for Expedited Forwarding (EF). Router implementations should treat the 5-bit PHB field as an index to be used in selecting a

particular packet handling mechanism. DiffServ flows are all for unidirectional traffic only. They are for traffic aggregates, not individual micro-flows

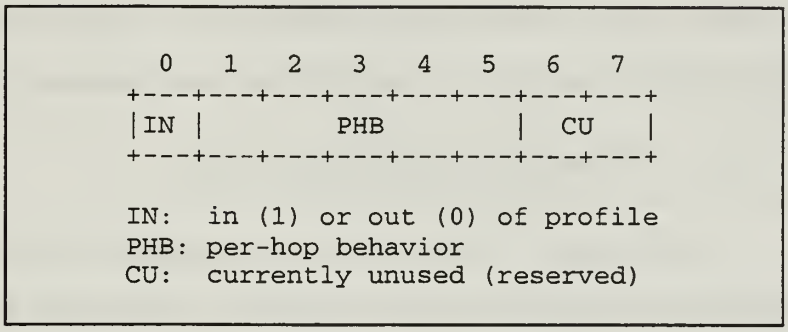


Figure 2.3 - Type of Service Field

a) Requirements of Differentiated Service

In order to receive differentiated services, the customer must have a service level agreement (SLA) with the service provider or ISP. A SLA is a profile (policing profile) describing the rate at which traffic can be submitted at each service level. Packets submitted in excess of this profile may not be allotted the service level requested. Each SLA has a Service Level Specification or SLS which defines the technical specification part of the contractual SLA.

SLSs may be static or dynamic. Static SLSs are the norm at the present time. They are instantiated as a result of negotiation between human agents representing provider and customer. A static SLS is first instantiated at the agreed upon service start date and may periodically be renegotiated (on the order of days or weeks or months). The SLS may specify that service levels change at certain times of day or certain days of the week, but the agreement itself remains static. A Dynamic SLS, on the other hand, may change frequently. Such changes may result for example, from variations in offered traffic load relative to preset thresholds or from changes in pricing offered by the provider as the traffic load fluctuates. A Dynamic SLS changes without human intervention and thus requires an automated agent and protocol, for example, a bandwidth broker to represent the differentiated service provider's domain.

An important subset of a SLS is the traffic conditioning specification or TCS, which specifies the detailed service parameters for each service level such as expected throughput, drop probability, latency. In addition to the details in the TCS, the SLS may specify more general service characteristics such availability/reliability, encryption services, routing constraints, authentication mechanisms, etc.

b) Operation of Differentiated Service

At the ingress of the ISP networks, packets are classified, policed, and possibly shaped according to the specification given in the SLS. The traffic classification and conditioning process is as depicted in Figure 2.4. If a packet traverse from one domain to another, its DS field may be remarked as determined by the SLS between the two domains.

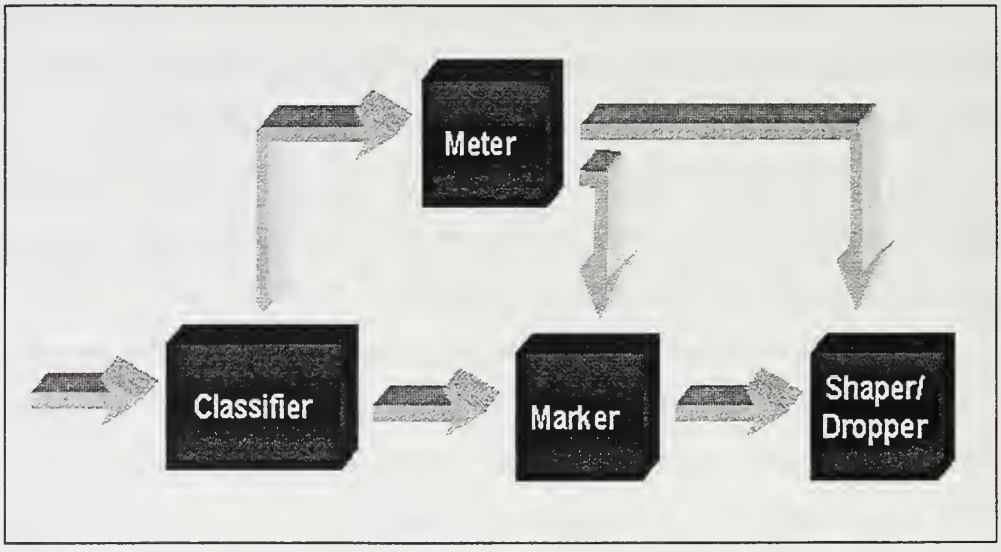


Figure 2.4 - Traffic classification and conditioning

Classifiers select packets based on some portion of their packet header and steer packets matching some classifier rule to another traffic conditioner for further processing. There are two types of classifiers: Multi-Field (MF) classifiers which can classify on the DS byte as well as any one of a number of header fields (like a RSVP classifier), or Behavior Aggregate (BA) classifiers which classify only on patterns in the DS byte.

Markers set the DS byte to a particular bit pattern, adding the marked packets to a particular differentiated services behavior aggregate.

Policers: monitor the dynamic behavior of the packets steered to them by a classifier and take an action (usually remarking or dropping packets) based on the relationship of measured properties of the packet stream to configured properties (e.g., rate and burst). Policers are generally placed after either type of classifier: after MF classifiers (e.g., at a host/network or site/provider boundary) or after BA classifiers (e.g., at a provider/provider boundary).

Shapers cause conformance to some configured traffic properties (e.g., token bucket). Like policers, shapers are generally placed after either type of classifier. Only one of the two primitives, policers or shapers, would be expected to appear in the same traffic

c) Types of Service Class

Currently, three types of service class has been identified for DiffServ:

- Premium Service. Premium Service is a peak limited, low delay service, resembling a leased line. It is for application requiring low-delay and low-jitter service. Possible application for such a service class are videoconference, fixed size transfer in fixed time, virtual leased line, and low delay applications.
- Assured Service. Assured Service .is characterized by a rate and burst profile. Application that may use this service class are those that need to transfer fixed file size in desired time or "better than best effort" applications
- Olympic Service. Olympic Service is further divided into three level of services - gold, silver and bronze; in decreasing order of congested link share. When encountering a congested link, packets with "Olympic gold" service will get a larger share of the link than packets sent using the "Olympic silver" service which gets a larger share of the link than packets sent using the "Olympic bronze" service.

d) Advantages of DiffServ over IntServ

DiffServ has only a limited number of service classes (due to the size of the DS field) as compared to IntServ. Consequently, the amount of state information, which is proportional to the number of classes, is much less than that for IntServ where the state information is proportional to the number of flows. This makes DiffServ more scalable than IntServ.

Deployment of DiffServ can be done in an incremental manner. DS-incapable routers will simply ignore the DS field of the packets and treat them as best-effort service.

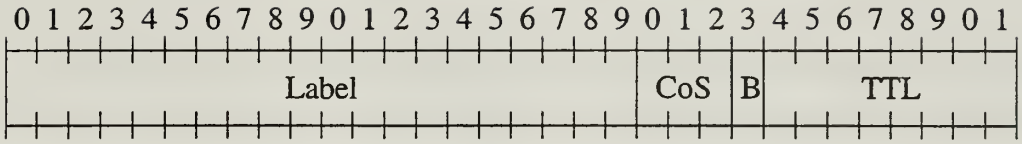
3. MultiProtocol Label Switching

Multiprotocol Label Switching, or *MPLS*, is a label-swapping, packet forwarding scheme evolved from Cisco's *Tag Switching* [6]. Classification, forwarding, and services for the packets are based on a fixed length label, which is appended in front of network protocol header. The network protocol may be IP or others, therefore MPLS is protocol independent. MPLS is very similar to DiffServ in that it also affects per hop behavior to provide QoS.

a) MPLS Operation

MPLS needs a Label Distribution Protocol (LDP) to distribute labels to set up label switched paths (LSPs). LSP setup may be control-driven (i.e., triggered by control traffic such as routing updates) or data-driven (i.e., triggered by the request of a flow). A forwarding table indexed by labels is constructed as a result of the label distribution. Each forwarding table entry specifies how a packet carrying the indexing label is to be processed.

Packets are labeled at the ingress or edge routers of a MPLS capable domain. Each MPLS packet has a 32-bit label header as shown in Figure 2.5.



Label : Label value, 20 bits
CoS : Class of Service, 3 bits
B : Bottom of Stack, 1 bit
TTL : Time to live, 8 bits

Figure 2.5 - MPLS Header

The MPLS header is encapsulated between the link layer header and the network layer header. An MPLS-capable router, called the label-switched router (LSR), examines only the label in forwarding the packet [6].

C. SIMILARITIES AND DIFFERENCES OF QOS MODELS

In summary, the similarities and differences of the three QoS models are tabulated in Table 1 below.

IntServ	DiffServ	MPLS
State info. is proportional to no. of flows which can be very large	State info. is proportional to no. of service class which is limited	State info. is proportional to no. of label switched paths which can be large
Resources are already reserved hence router's work is minimum	Most of the work occurs at the border router	Label is assigned by and switched at transit routers
Router must be IntServ capable	Router need not be DiffServ capable	Router must be MPLS capable
Require resource reservation	No resource reservation needed	No resource reservation needed
IP Protocol only	IP Protocol only	Any network protocol

Table 1 - Similarities and Differences of QoS Models

THIS PAGE INTENTIONALLY LEFT BLANK

III. SAAM QOS MANAGEMENT DESIGN

Two categories of service classes have been defined for the Internet: 1) Integrated Service and 2) Differentiated Service. MPLS, as mentioned in previous chapters, is a form of Differentiated Service. The SAAM server is designed to support all these services and it will deploy the necessary functionality to the SAAM routers. The coexistence of these services is possible by partitioning a link into different logical service level pipes [15] and assigns them to different services.

The SAAM server needs to coordinate resource allocations among different QoS services. It performs two levels of link bandwidth allocation. At the top level, the server must allocate bandwidth to each service level pipe. At the next level, the server must allocate bandwidth to individual flows or customers that share one service level pipe.

There are actually more network resources than just link bandwidth (e.g., buffer space and CPU time), however because of time constraints, this thesis is focused only on link bandwidth. There are also more QoS requirements than just throughput (e.g., queueing delay and loss rate). In this thesis, we assume that appropriate packet scheduling algorithms and admission control criteria will be used to ensure that other QoS requirements of a flow will be met if sufficient bandwidth is allocated to the flow. One may ask how much is sufficient. That is the role played by the Alpha parameter in our admission control equations.

In order to optimize the use of network resources under varying network conditions, the SAAM server should employ an adaptive QoS routing algorithm. Adaptive QoS routing refers to the ability to dynamically switch to a new QoS routing algorithm, such as Shortest-Widest Path [12], Widest Shortest Path [12] or Short-Distance Path [12], based on the current network conditions. QoS routing that satisfies multiple constraints have been proved to be NP-complete. [12]

A. NEW MESSAGES REQUIRED

In the previous chapter, we pointed out that IntServ and dynamic SLS of DiffServ need a reservation protocol to request and reserve resources. In SAAM, a user or

application requests for resources and the server makes reservation of the required resources. A reservation protocol is needed to facilitate communications between the two entities. The SAAM server will send *ResourceAllocation* messages to those routers under its control to allocate trunk capacity to various SLPs.

An application requiring IntServ or dynamic SLS at an end host will trigger a request through an edge router, which functions as a bridge between the application and the server. The request is forwarded from the edge router to the server in a *FlowRequest* message. Upon receiving the message, the server performs admission control for the request, trying to find a path that can support the QoS parameters and resource requirements encoded in the request message. The server then informs the edge router of the result (i.e. acceptance or rejection, etc) by sending it a *FlowResponse* message. If the request for an IntServ flow is accepted, the response message will contain a *flow id* that the packets of this flow should carry in their header. When the application is done with the flow, it may trigger the edge router to send a *FlowTermination* message to the server to explicitly request the server to release the resouces allocated to the flow.

1. SAAMPacket Format

The packet format used in the SAAM emulator [15] is shown in Figure 3.1. The original payload structure (third row in the figure) is inherently inefficient. Its *Type Id* can only have a value of 0 for SAAM messages or 1 for Resident Agent class file. The new payload structure has less overhead and has a *Type Id* that ranges between 1 to 16 - see Appendix H for all SAAM message types in the *saam.message.message* class. Note that the original payload structure is still supported for backward compatibility.

SAAMPacket

8	Varied
Header	Payload

SAAMHeader

8	1
Time Stamp	#of Messages

Original structure of payload portion

1	1	Name Length	2	Bytecode Length
Type_Id	Name Length	Name of Classfile	Bytecode Length	Bytecode

New structure of payload portion

1	2	Bytecode length
Type_Id	Bytecode length	Bytecode

Figure 3.1 - SAAM Packet Structure

2. ResourceAllocation Message

In order for the server to have full control over the resources available in the region, it sends *ResourceAllocation* messages to the routers under its control to initialize their trunk allocations as soon as the control channel to its routers has been established. The server also creates a Path Information Base (PIB) [15] based on initial feedbacks from the routers.

Figure 3.2 shows an example of a *ResourceAllocation* message for five service levels. This message contains three fields. The first field is the *Type_Id* for this message (which is 16). The second is the *bytecode length* of the bytecode that follows in the third field. The bytecode is made of an array of 4-byte integers, specifying the amount of bandwidth allotment (in Kbps) for each service level. In this example, the second field contains the value of 20, which is equivalent to the number of service level pipes (five for this example) multiplied by four.

1	2	20
Type_Id	Bytecode Length	Bytecode
16	5*4 = 20	Service Level Allotment Parameters for each SLP

Figure 3.2 - Resource Allocation Message

For this thesis, the initial allotments used are as follows:

- $0.1B_{max}$[1] for SLP[2] of Control flows (SAAM messages)
- $0.4B_{max}$ for SLP of IntServ flows
- $0.3B_{max}$ for SLP of DiffServ flows
- $0.2B_{max}$ for SLP of Best Effort flows
- 0 for SLP of others (e.g. tagged packets, etc)

These amounts are determined based on the following assumptions:

- Control traffic consumes less than $0.1B_{max}$.
- IntServ flows may be charged more than others because of their guaranteed performance.
- DiffServ flows may be charged more than best effort traffic.
- ISPs do not want starvation of the best effort service.
- Tagged packets should have the lowest priority and may be transmitted only if other SLPs are idle.

As the network resources are utilized, there may be a need to adjust the current resource allotments given to the various SLPs. The conditions that would trigger such adjustments will be discussed later. If the packet scheduler employed at the routers is rate based (e.g., *Weighted Fair Queueing, Virtual Clock, Self-Clocked Fair Queueing*, etc), then the server will need to send new *ResourceAllocation* messages to all the routers when adjustments are made. Otherwise, if the packet scheduler employed is priority

[1] B_{max} is the total link bandwidth.
[2] SLP is the acronym for Service Level Pipe.

based, no *ResourceAllocation* message will be send to the router as packets with the highest priority will be serviced first.

3. FlowRequest Message

After allocating resources to various service level pipes at each link, the server is ready to receive and process flow requests. When a flow request is received, the server needs to determine which service the request is for. The existing flow request message does not provide sufficient information for the server to do so [15]. Hence a new flow request message format needs to be in place. The new *FlowRequest* message format includes a service level field. However, the requirements for an IntServ flow would differ from that of a DiffServ flow. A DiffServ flow request should contain the *User id* of the requestor and the requested ServiceLevelSpec (*SLS*) instead of the delay, loss rate and throughput requirements (see Figure 3.3 and 3.4 respectively).

1	2	16	16	8	1	4	7 - 11
Type	Bytecode Length	Source	Destination	Time Stamp	Service Level	User Id	ServiceLevelSpec
9					1 = Contorl 2 = IntServ 3 = DiffServ 4 = BestEffort 5 = Others		

Figure 3.3 - Flow Request for DiffServ flow

1	2	16	16	8	1	4	4	4
Type	Bytecode Length	Source	Destination	Time Stamp	Service Level	Delay	Loss Rate	Throughput

Figure 3.4 - Flow Request for IntServ flow

1	4	1	1/2/5
DSCP	Profile	Scope	Disposition of non-conforming traffic

Figure 3.5 - Service Level Spec Parameters

The IETF Internet Draft on DiffServ recommends a service level specification (see Figure 3.5) to have a minimum of four fields. The first field is the differentiated service code point or *DSCP* (see Figure 3.6) that will be used to tag the packets for a specific DiffServ service class. Bit '0' of the DSCP indicates whether the packet is in or out of the profile specified for it. Bit 1 to 5 is used for per hop behavior (PHB). Bit '1' is for delay priority packet. Bit '2' is for throughput priority packet. Bit '3' is for loss rate priority (i.e. minimum loss rate). Bit 6 and 7 are currently not used.

Bit Position							
0	1	2	3	4	5	6	7
IN	PHB					CU	

1 = in
0 = out

Figure 3.6 - Differentiated Service Code Point Format

The second field contains the *Profile* that specifies the amount of throughput (in Kbps) that this service class is allowed to consume. The third field specifies the *Scope* to which this service level spec is applicable. The last field defines the disposition action that is to be taken when the profile given is exceeded by the actual flow traffic (see Figure 3.7). If the action is to discard, then no other parameter is necessary. If the action is to remark, then the new DSCP to be used will be required. If the action is to reshape, then the shaping profile to be used will need to follow (e.g. shaping the profile to 200Kbps from 500Kbps).

1	0/1/4
Action	-/New DSCP/Shaping Profile

1 - Discard
2 - Remark
3 - Shape

Figure 3.7 - Disposition Action Parameters

4. FlowResponse Message

When the server receives an IntServ or DiffServ flow request, it will execute the respective admission control and sends a flow response back to the edge router that forwarded the request. The flow response message will notify the router the result of the flow request along with other information that may be required. Figure 3.10 shows the *FlowResponse* message format. The first field is the *Type_Id* for this message (which is 8). The second is the *Result* field that contains the result of the admission control carried out by the server. If the result is an acceptance of a DiffServ flow request, the next field will contain the *SLS* allocated for it. If the result is an acceptance of an IntServ flow request, the *Flow_Id allocated to the new flow* will follow.

1	2	8	1	7 - 11 / 3
Type_Id	Bytecode Length	Time Stamp	Result	SLS / Flow_Id
8			0 – Service Unknown 1 - IS Accepted 2 - DS Accepted 3 – Negotiated 4 – Unreachable 5 - SLA not available	

Figure 3.8 - FlowResponse Message Format

5. FlowTermination Message

When an application is done with the flow assigned to it, it will trigger the edge router to send a *FlowTermiination* message to the server so that the server can update its PIB and releases the resources that have been allocated to the flow. Figure 3.9 shows the *FlowTermination* message format. Note that a bytecode length field is not required for this case as the message size is fixed.

1	3
Type_ID	Flow_Id
12	

Figure 3.9 - FlowTermination Message Format

6. Management of Service Level Spec

After the resources have been pre-allocated, the server - which stores and maintains information for all the DiffServ customers including their respective Service Level Agreements (SLAs) - will need to send the SLS Tables to all edge routers. The edge routers then store this SLS Table for admission control and policing of customers who have signed up for *DiffServ*.

The mechanism for the server to do that is by sending *SLSTableEntry* messages to the edge routers. Figure 3.10 shows the *SLSTableEntry* message format. The first field is the *Type_Id* for this message (which is 17). The second is a *User_Id*. The third field contains the *SLS* to be used for the customer with the *User_Id*.

1	2	4	7 – 11/4
Type_Id	Bytecode Length	User_Id	SLS / Node_Id
17			

Figure 3.10 - SLSTableEntry Message Format

The server needs to know when the user is done with the dynamic SLS allocated. The router sends a *SLSTableEntry* message that contains a *Node_Id* as its third field to the server, which uses it to identify the user who has no need of the SLS assigned to him any more. Hence the server will update its SLS database by deleting the user from the SLS Table of the node with that *Node_Id*.

B. SAAM QOS MANAGEMENT

The flow chart that describes SAAM's QoS management process is shown in Figure 3.11. The function of the QoS management component may be broken down into several parts:

- Resource management
- Path selection with adaptive QoS routing
- Processing of IntServ flow.
- Processing of DiffServ flow.

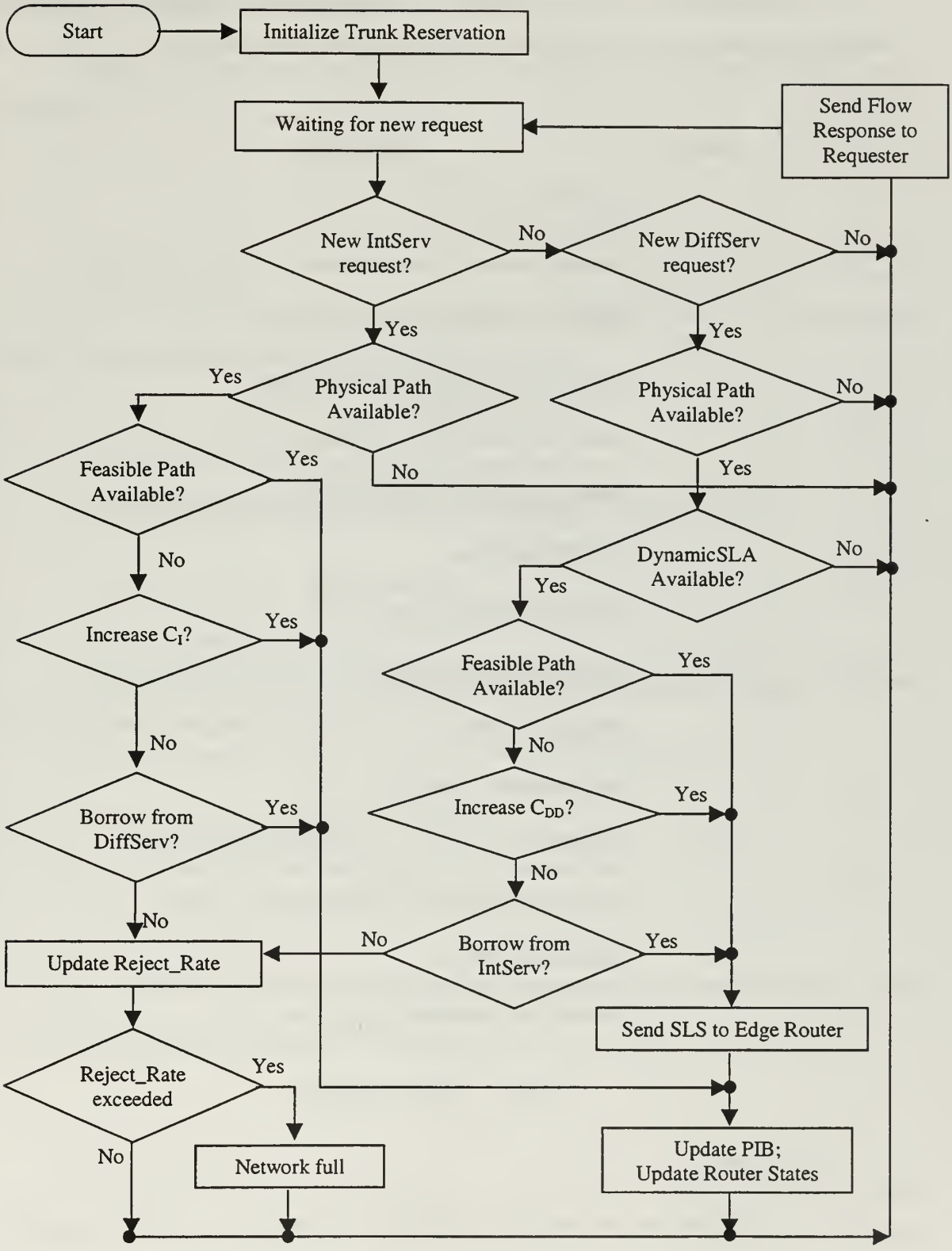


Figure 3.11 – Flow Chart of SAAM QoS Management Algorithm

Throughout the rest of this chapter, the various QoS management functions will be discussed and presented using the following mathematical notations.

Symbol	Definition
B	Set of active flows
α	Loading factor of a service class
β	Expanding factor of a service class
ρ	Borrowing factor of a service class
C	Trunk allocation of a service class
R	Bandwidth requirement of a flow
f	An arbitrary flow
f^*	The flow being requested
CC	SAAM Control Channel
I	Integrated Service
D	Differentiated Service
DD	Dynamic SLA portion of Differentiated Service
DS	Static SLA portion of Differentiated Service
BF	Best Effort Service
N	Number of customers/applications

Table 2 - Table of Symbols

For example,

B_I denotes the set of active flows for IntServ

C_{DD} denotes the trunk allocation for the DiffServ Dynamic SLA service.

1. Resource Management

Suppose the maximum trunk capacity of the link in consideration is C_{max}. Then we have

$$C_I + C_D + C_{CC} + C_{BF} \leq C_{\max} \quad (1)$$

Assume that $0.1C_{max}$ is allocated to the SAAM control channel and $0.2C_{max}$ is allocated to the best effort service as its minimum share of the link. Then the maximum bandwidth that may be allocated to guaranteed service is

$$C_I + C_D \leq 0.7C_{\max} \quad (2)$$

The maximum throughput that may be allocated for IntServ flows is

$$\sum_{f \in B_I} R_f \leq \alpha_I C_I \quad (3)$$

The maximum throughput that may be allocated for Static and Dynamic DiffServ flows are given by equation 4 and 5 respectively.

$$\sum_{f \in B_{DS}} R_f \leq \alpha_{DS} C_{DS} \quad (4)$$

$$\sum_{f \in B_{DD}} R_f \leq \alpha_{DD} C_{DD} \quad (5)$$

C_{DS} is determined a priori to exactly meet the requirements of the *DS* flows, and the admission criteria for it is given by equation 6.

$$N_{DS} R_{DS} = \alpha_{DS} C_{DS} \quad (6)$$

For example, if C_{max} = 100Mbps, then C_I may be allocated an initial minimum throughput of $0.2C_{max}$ or 20Mbps. This will allow for twenty IntServ flows of 1Mbps each. Similarly, C_D may be given an initial minimum throughput of $0.2C_{max}$, where C_{DS} and C_{DD} each get $0.1C_{max}$. Hence the whole trunk capacity may be viewed as a pie chart

illustrated in Figure 3.12. As the utilization of the network progresses, the unallocated throughput, which is $0.3C_{max,}$ may be dynamically allocated to C_I or C_D. Note that C_I and C_D could increase until equilibrium is reached where there is no unallocated throughput.

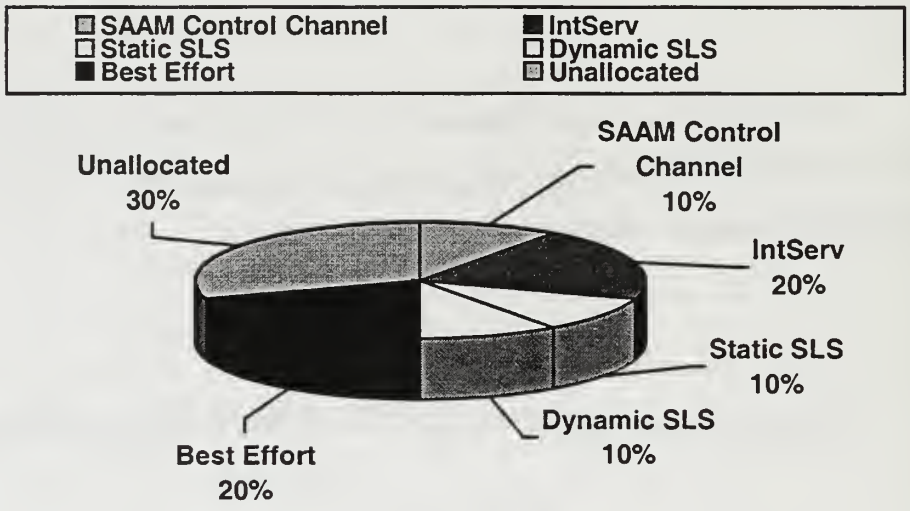


Figure 3.12 - Pie Chart illustration of SLP resource allocation

2. Processing of IntServ Flows

After identifying the type of service request to be a IntServ flow request, the server will look for all the possible physical paths that are able to reach the destination requested. If no path is available, then the server will notify the client that the destination is unreacheable.

If the destination is reacheable, the server will look for the best path that is able to meet the QoS requested such that

$$\sum_{f \in B_I} R_f + R_{f^*} \leq \alpha_I C_I \qquad (7)$$

If the best path is not able to meet the QoS requested, the server will check if the request may be met by increasing C_I by a small factor of β_I such that,

$$\sum_{f \in B_I} R_f + R_{f^*} \leq (1+\beta_I)\alpha_I C_I \qquad (8)$$

where the typical value for β_I derived from equation 8 is given by,

$$\beta_I = \frac{\sum_{f \in B_I} R_f + R_{f^*}}{\alpha_I C_I} - 1 \qquad (9)$$

If C_I cannot be increased this way because there is no sufficient unallocatated bandwith, the server will check if inter-service borrowing may be carried out such that the request may be met by borrowing some bandwidth from the *DS* portion of DiffServ. The new admission control condition becomes

$$\sum_{f \in B_I} R_f + R_{f^*} \leq \alpha_I (C_I + \rho_{DD} C_{DD}) \qquad (10)$$

where $\rho_{DD} C_{DD}$ specifies the maximum amount that may be borrowed and ρ_{DD} is given by

$$\rho_{DD} = 1 - \frac{\sum_{f \in B_{DD}} R_f + 0.2 \sum_{f \in B_{DD}} R_f}{\alpha_{DD} C_{DD}} \qquad (11)$$

Equation 11 is based on the statistical theory of confidence intervals, which states that the probability of the population mean between two bounds, e.g. k_1 and k_2, is given by the confident coefficient *1-x*, where *x* is the significance level [18]. For our case, we assume that the throughput of the *DD* flows in the near future follows a normal distribution with a mean of $\sum_{f \in B_{DD}} R_f$ and variance of σ^2. Consider the confidence interval of $\left[0, \sum_{f \in B_{DD}} R_f + 2\sigma\right]$ (see Figure 3.13). From the Table A.2 and Table A.3 of [18] on the quintiles of the unit normal distribution, the throughput of *DD* flows in the near future will fall in the interval of $\left[\sum_{f \in B_{DD}} R_f - 2\sigma, \sum_{f \in B_{DD}} R_f + 2\sigma\right]$ with a probability of 0.9546.

Consequently, the throughput will fall in the interval of $\left[0, \sum_{f \in B_{DD}} R_f + 2\sigma\right]$ with an even higher probability of $(1 - (1 - 0.9546) / 2) = 0.9773$. Therefore, we set the boundary for borrowing to be $\sum_{f \in B_{DD}} R_f + 2\sigma$ (see Figure 3.13). Furthermore, since the number of active *DD* flows is large, σ should be small relative to the mean. We assume that $\sigma = 0.1 \sum_{f \in B_{DD}} R_f$, which leads to equation 11 for determining ρ_{DD}.

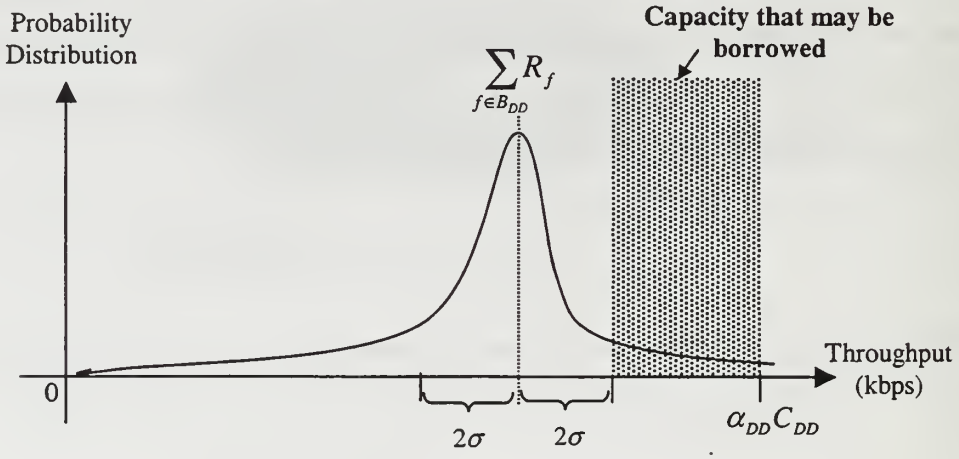


Figure 3.13 - Statistical Distribution of the aggregate throughput of DD flows

If Inter-Service borrowing cannot be done, then the server will compute the new rejection rate for the service requested. If the rejection rate exceeds a specified rejection threshold, the server will make a log of the event to indicate that the network may need to be upgraded.

3. Processing of DiffServ Flows

After identifying the type of service request to be a DiffServ flow request, the server will look for all the possible physical paths that are able to reach the destination requested. If no path is available, then the server will notify the client that the destination is unreacheable.

If the destination is reacheable, the server will look for the best path that is able to meet the QoS requested to admit a new dynamic SLA such that

$$\sum_{f \in B_{DD}} R_f + R_{f*} \leq \alpha_{DD} C_{DD} \qquad (12)$$

If the best path is not able to meet the QoS requested, the server will check if the request may be met by increasing C_{DD} by a small factor of β_{DD} such that,

$$\sum_{f \in B_{DD}} R_f + R_{f*} \leq (1+\beta_{DD})\alpha_{DD} C_{DD} \qquad (13)$$

where the typical value for β_{DD} derived from equation 13 is given by

$$\beta_{DD} = \frac{\sum_{f \in B_{DD}} R_f + R_{f*}}{\alpha_{DD} C_{DD}} - 1 \qquad (14)$$

If C_{DD} cannot be increased due to insufficient unallocated bandwidth, the server will check if inter-service borrowing may be carried out such that the request may be met using the admission condition given by equation 15.

$$\sum_{f \in B_{DD}} R_f + R_{f*} \leq \alpha_{DD}(\rho_I C_I + C_{DD}) \qquad (15)$$

where by the same argument used to derive equation 11, ρ_I is given by

$$\rho_I = 1 - \frac{\sum_{f \in B_{DD}} R_f + 0.2 \sum_{f \in B_{DD}} R_f}{\alpha_I C_I} \qquad (16)$$

If inter-service borrowing cannot be done, then the server will compute the new rejection rate for the service requested. If the rejection rate exceeds specified rejection threshold, the server will make a log of the event to indicate that the network may need to be upgraded.

When a *DD* flow is admitted, the associated SLS is sent to the edge router for policing and shaping of the DiffServ flow.

4. Path Selection With Adaptive Routing

Path selection refers to choosing the "best" path among the set of feasible paths that are able to support the QoS parameters specified in a flow request. The path selection algorithm in the current SAAM prototyped server code will be reused as much as possible. However, the algorithm only selects the first possible path found in the PIB. Hence, the algorithm for QoS Routing [12] will be added.

As mentioned in Chapter 2, there are three main types of QoS routing schemes: Widest-Shortest Path (WSP), Shortest-Widest Path (SWP), and Shortest-Distance Path (SDP) [12]. In order for the server to select the appropriate routing scheme, it needs to monitor the network resources, e.g. the packet loss rate and delay, queue length, etc, on an interface. A high rate of dropped packets is an indication of a bottlenecked link; so does a large queue length.

SWP will be the default scheme since it emphasizes on preserving network resources by choosing the shortest paths first. WSP will be the choice of scheme when the load in the network is unbalanced since it emphasizes load balancing by choosing the widest paths first. As SDP makes a trade-off between the WSP and SWP, it will be left for future investigation.

IV. SAAM QOS MANAGEMENT IMPLEMENTATION

A Java based SAAM server and router prototype has been developed by previous graduates. In this chapter, the various additions and modifications to the prototype, pertaining to QoS management, will be discussed. Note that there will be many references to the previous chapter.

A. NEW MESSAGES

A *ResourceAllocation* class, a *FlowTermination* class and an *SLSTableEntry* class are added to *saam.message* package. The existing *FlowRequest* and *FlowResponse* classes are modified to implement the new message format described in the previous chapter.

1. *ResourceAllocation* Class

The *ResourceAllocation* class is an extension of the *Message* abstract class (see Appendix H). This class is used by the server to instantiate a *ResourceAllocation* message object. The message is sent to every router to allocate resources for various service level pipes at each of its outgoing links (interfaces) (see Appendix C).

2. *FlowRequest* Class

The current *FlowRequest* class is modified to include constructors and all the necessary data members for IntServ and DiffServ flow requests (see Appendix D).

3. *FlowResponse* Class

The current *FlowResponse* class is modified to include constructors and all the necessary data members for IntServ and DiffServ flow responses (see Appendix E).

4. *FlowTermination* Class

A new *FlowTermination* class is created under the *saam.message* package. This class extends the abstract *Message* class (see Appendix F).

5. *SLSTableEntry* Class

A new *SLSTableEntry* class is created under the *saam.message* package. This class extends the abstract *Message* class (see Appendix G).

6. *PacketFactory* Class

The *processPacket( )* method and the *append(Message me)* method have been modified to handle the new types of messages (see Appendix L).

7. *ControlExecutive* Class

The *requestFlow( )* method has been modified to handle the new types of *FlowRequest* messages (see Appendix K). A new *processMessage(byte[] bytes, String message)* method is added to process all the new messages.

B. RESOURCE MANAGEMENT

1. *Server* Class

The methods added to this class (see Appendix B) are:

a) public void initializeResourceAllocation(IPv6Address address)

This method is used to send a resource allocation message to a router with the IPv6Address specified in the parameter to initialize the amount of resources allocated to its service level pipes.

b) public void sendResourceAllocation(IPv6Address destination, int[] allocated_throughput)

This method is used to send a resource allocation message to the router with the specified *destination* address to update the amount of resources it has been allocated for its service level pipes.

2. *ClassObjectStructure* Class

The methods added to this class (see Appendix I) are:

a) public void updateEffectiveQoSOfPath(int path_id, int effectiveDelay, int effectiveLossRate, int effectiveThroughputRemaining)

This method is used to update the effective QoS parameters of a path.

C. PATH SELECTION WITH ADAPTIVE QOS ROUTING

Path selection in QoS Management refers to finding a feasible path that is able to support the QoS parameters specified in a flow request. The *ClassObjectStructure* class currently has a method called *getPathThatCanSupportFlowRequest(int source_router, int destination_router, FlowRequest myFlowRequest)* that returns the *path_id* of a path that is able to support the QoS parameters specified in *myFlowRequest*. However, for backward compatibility, a new method called *getPathThatSupportFlowRequest(int source_router, int destination_router, FlowRequest myFlowRequest)* will be added to the *ClassObjectStructure* class to handle the two new types of flow requests.

As mentioned in Chapter 2, there are three main types of QoS routing schemes: Widest-Shortest Path (WSP), Shortest-Widest Path (SWP), and Shortest-Distance Path (SDP). Since the shortest path and widest path are easily obtainable from the current PIB implementation in SAAM, the selection between Widest-Shortest Path and Shortest-Widest Path shall be implemented first.

1. *ClassObjectStructure* Class

The methods added to this class (see Appendix I) are:

a) public int getPathThatSupportFlowRequest(int source_router, int destination_router, FlowRequest myFlowRequest)

This method determines if there is a path that can support a particular flow request. A returns value of -1 mean the destination requested for is not reachable. A return value of zero indicates that the destination is reachable but no path can support the QoS parameters requested.

b) private int determineBestPath(Path newPath, int newPathId, Path bestPath, int bestPathId)

This method returns the *path_id* of the best path between *newPath* and previous *bestPath* with the type of routing algorithm used (WSP or SWP).

c) public int getRemainingThroughput(IPv6Address address, byte service_level)

This method returns the remaining throughput of an interface with the *address* and *service_level* specified in the parameters.

d) public IPv6Address[] getPathAddress(int path_id)

This method returns an array of interfaces' address that forms a path the the *path_id* in the parameter.

e) public Hashtable getAllPossiblePaths(int source_router, int destination_router)

This method returns a hashtable of all possible paths for the *source_router* and *destination_router* specified in the parameters.

D. MANAGEMENT OF FLOWS

In SAAM, routers send flow requests to the server, while the server carries out admission control and sends flow responses back to the routers. When a message

received at the server node is identified to be a flow request, the ControlExecutive will create an instance of it and call the *processFlowRequest( )* method of the *Server* class. If the flow request message is identified as one for IntServ, the server will execute the admission control process *IS_Admission( )* in the *Server* class. If the flow request is identified as one for DiffServ, it will execute the admission control process *DS_Admission( )*. If neither is the case, the server will process the flow request before for backward compatibility.

1. *Server* Class

Two new methods are added to this class (see Appendix B)

a) private void DS_Admission(int source, int destination, FlowRequest flow_request)

This method is carries out the admission control for a DiffServ flow request. If a request is accepted, a new SLS will be created and added to the SLS database. The *sendSLSEntry( )* method will then be called to install the flow state at the edge router. In all cases, the server will call *sendFlowResponse( )* to notify the edge router of the outcome of the admission control.

b) private void IS_Admission(int source, int destination, FlowRequest flow_request)

This method carries out the admission control for an IntServ flow request. If a request is accepted, a call to *updateRouter( ) will be made to install the necessary state information for the new flow to each router on the flow path.*

c) public void receiveFlowTermination(int flow_id)

This method is called when the router receives a *FlowTermination* message to remove the *flow_id* specified in the parameter from the PIB.

2. *ClassObjectStructure* Class

A method is added to this class.

a) public void deleteAssignedFlow(int flow_id){

This method is called by *receiveFlowTermination(int flow_id)* to remove the *flow_id* specified in the parameter from the PIB.

E. MANAGEMENT OF SERVICE LEVEL SPEC

A new package is created for the purpose of supporting DiffServ. The new package is located under *saam.server* named *diffserv* (see Appendix J). It is comprised of three classes.

1. *SLS* Class

This class is used to store information according to the specification required for a service level spec (*SLS*).

2. *SLSTable* Class

This class extends the Java *HashTable* class to create a hash table of *SLS* objects. It is used by every edge router to keep track of all the *SLS*s assigned for various customers that use the router as the entrance to the network.

3. *SLSDbase* Class

This class extends the Java *HashTable* class to create a hash table of *SLSTable* objects. It is maintained and used by the server to keep track of all the *SLS*s assigned for various customers.

In addition to *saam.server.diffserv* package, a new FileIO class has been added to the *saam.util* package (see Appendix N) and new methods have been added to the server class of *saam.server* package (see Appendix B).

4. *FileIO* Class

This class provides the methods to read/write to a file.

5. *Server* Class

The new methods added (see Appendix B) are:

a) private void setupSLSDbase( )

After the server has been instantiated, it then calls this method to set up its *SLS* database.

b) private void addSLSTable(StringTokenizer st)

This method is called by *setupSLSDbase( )* to set up its *SLS* database.

c) private boolean SLA_available(int sourceNode, int throughput)

This method is used to check if the resources at the *sourceNode* is sufficient to support new *SLS*.

d) private void sendSLSTable(IPv6Address routerId, Integer nodeID)

This method is used to send SLSTable that is associated to the router specified in the parameters.

e) private void sendSLSTableEntry(IPv6Address routerId, int user_id, SLS sls)

This method is used to send a *SLSTableEntry* message that contains the *user_id* and *SLS* to the specified in the parameters.

f) ***public void receiveSLSTableUpdate(SLSTableEntry message)***

This method is called when the router receives a *SLSTableEntry* message that contains the *user_id* and *SLS* from the server.

g) ***private SLS addSLS(Integer source, FlowRequest flow_request)***

This method is used to add a *SLS* to the *SLSDbase* when a new *SLS* can be admitted.

h) ***private void deleteSLS(Integer node_id, int user_id)***

This method is used to delete or remove a *SLS* from the *SLSDbase* when a *SLSTableEntry* message is identified as a *SLS* withdrawal message type..

F. INTERSERVICE RESOURCE BORROWING

Inter-service resource borrowing between IntServ and DiffServ is a difficult subject that has never been fully studied before. As an initial step, only IntServ will be allowed to borrow resources from DiffServ.

V. TEST AND VERIFICATION

As the focus of this thesis is on the server, it will be more efficient to limit the test environment locally at the server. This will also make it possible to verify the test results obtained. However, in order to verify the correctness of processing the new message types, a simple test topology of a server and at least one router is required.

A. MESSAGE VERIFICATION TEST

1. Test Requirements

The simple test topology that we used is shown in Figure 5.1, where the number in square brackets is the MAC address of the interface while the numbers to its left is the IPv6Address of that interface. A *Demo_1Server_1Router* class (see Appendix O) is created to function as the demo station that stands up the server and router according to the test topology. A *SendFlowAgent* class (see Appendix P) is then used to send the required resident agents to both the router and server nodes. After the resident agents have been installed at the routers, the router designated as the sender will send a flow request to the server. The server responds with a flow response. The result is verified by comparing it to the expected flow response result. The server also sends other router-bound messages (i.e. *SLSTableEntry* and *ResourceAllocation*) to the router, and the router sends server-bound messages (i.e. *SLSTableEntry* and *FlowTermination*) to the server to test if these messages are handled correctly.

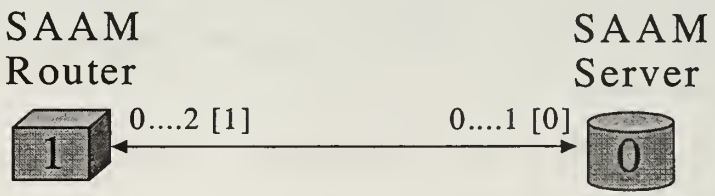


Figure 5.1 – One Server and One Router Topology

2. Test Results

Several problems were encountered during the initial testing. These were largely due to programming error like mishandling of packet format or error in appending packet information. After the teething problems were resolved, the message were verified to be correctly received and processed by both the server and the router.

B. QOS MANAGEMENT ALGORITHM TEST

1. Test Requirements

A server and three router topology shown in Figure 5.2 is used. This test topology is simple and yet adequate to test our QoS management algorithm.

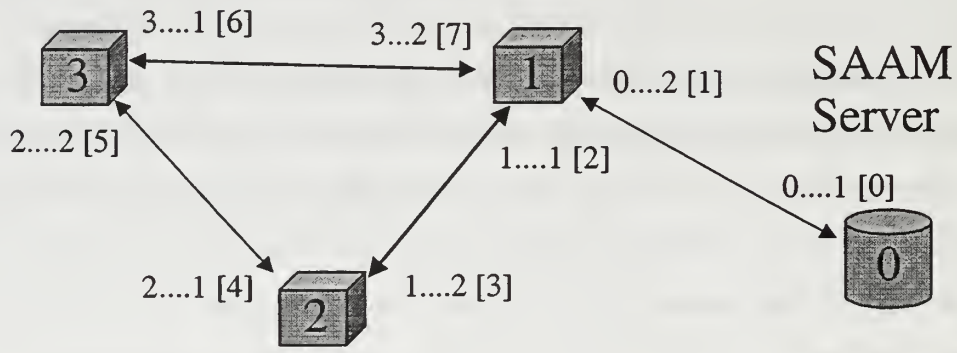


Figure 5.2 - One Server and Three Router Topology

A new packet called server.demo.QoSDemo package (see Appendix Q) that consists of four classes have been developed to setup and carry out the test sequence.

- *QoSDemo* class

  This is the main class that instantiate a FourNodes object to set up the topology and run the test.

- *FourNodes* class

  This class sets up the interfaces for the four nodes and creates four *NodeThreads* object (one for each node) that sends *Hello* messages [15] to set up the test topology.

- *NodeThread* class

  This class is responsible for generating flow requesters using the *RequesterThread* class. It waits for a response and verify the result.

- *RequesterThread* class

  This is a threaded class that will generate a request or a series of requests. It will wait for a response for each request sent and process it depending on the result contained in the flow response received.

The *processHello( )* method of the *server* class is reused to build up the PIB for the test. No actual sending and receiving of messages will be done in this local host test. This is to eliminate any possible message handling errors that may affect the testing, since the primary goal of this test is to verify the QoS managemetn algorithm. To do so, the test will start from the *processflowrequest( )* method of the *server* class, to simulate that the server have received a flow request and will be carrying out the respective admission control for the type of flow request received. Hence if the message handling have been verified to be in order, the success of this test means that the whole QoS management process will function correctly.

2. Test Results

It has been verified that the server is able to build up the PIB correctly as before with the Hello messages received. The admission control for *IntServ* and *DiffServ* are processed respectively to the type of service request. The resource allocation are in order and the requests are admitted according to the condition laid done in the previous chapters.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

An efficient QoS management algorithm has been implemented into SAAM. Our tests showed that the SAAM Server is now able to adapt its routing algorithm under varying network conditions.

A. LESSIONS LEARNED

SAAM is a huge project. It is developed by people from very different technological background. It has a great potential to be deployed in a few years time. A lot of lessons have been learned from its test and development. Some of the lessons learned are highlighted here.

1. Working With Large Project

Development of such a huge project like SAAM requires much coordination and cooperation from its project mates. Mutual encouragement and support have contributed much to its success despite the many difficulties encountered. The professor, being the leader of the team, has played a vital role of keeping the team in track and providing the necessary support and motivation.

2. Requirement Of Powerful Server

As more and more capabilities are added to SAAM server, the need for the server to be installed on a powerful PC has been more and more necessary. It is recommended that server code be deployed on faster machine in order to have a reasonable test bed.

B. FUTURE WORK

The SAAM prototype has come a long way to the current functional system. However, there are still several areas that may be improved upon.

1. Scheduler Capabilities At The Router

The current version of SAAM router does not have the scheduler algorithm to support the resource allocation and reservation carried out by the server. Adding a rate-based scheduler (e.g., WFQ or Virtual Clock) to the priority-based scheduler currently deployed will enable it to do so.

2. A Bridge Between The Customer And The Server

The SAAM router needs to have the capability to translate user/application requirements to an equivalent type of flow request for the server. This should be provided to the user/host in a transparent manner.

3. Security

In any commercial system, the importance of its security has been well highlighted in the recent attempts to flood network providers like YAHOO!, CNN, etc. in February 2000. Likewise, in order for SAAM to be accepted for deployment, it needs to be secured. SAAM server must be secured enough to prevent hackers from terminating its services. Its PIB must be well guarded from malicious attack such as illegal alterations. Similarly, its SLS database must be well guarded.

4. Fault Recovery

When the SAAM primary server goes down, the backup server needs to be able to take over control and reassign those flows affected. Currently, the backup server is only able to detect failure of the primary server.

5. Re-routing Of Flows During Interface Failure

The SAAM server needs to be able to re-route flows that may be affected by a link failure. This feature is currently not available.

APPENDIX A - CURRENT INTERNET ROUTING PROTOCOL

It is clear that some form of QoS Routing is required to provide quality of service in the Internet. Generally, there are two main category of Internet routing protocol: Best Effort Routing and QoS routing. Some popular best effort routing and QoS routing developed are presented here.

There are several types of best effort routing protocol being developed over the years. Examples of these protocols include: the Interior Gateway Routing Protocol (IGRP), the Enhanced Interior Gateway Routing Protocol (EIGRP), the Open Shortest Path First (OSPF) protocol, the Exterior Gateway Protocol (EGP), the Border Gateway Protocol (BGP), the OSI Routing protocol, the Advanced Peer-to-Peer Networking protocol, the Intermediate System to Intermediate System (IS-IS) protocol, and the Routing Information Protocol (RIP). Among these, the common ones are the RIP, OSPF and BGP. The later is used as a Interdomain Routing Protocol while the other two are used as a Intradomain Routing Protocol.

A. ROUTING INFORMATION PROTOCOL (RIP)

Currently there are RIP version 1 (RIPv1) and RIP version 2 (RIPv2) protocols running in the Internet. RIPv1 was one of the first dynamic routing protocols used in the Internet. It was developed as a technique for passing around network reachability information of relatively simple topologies. In RIP, routing information is passed between routers using the User Datagram (UDP) transport protocol.

Routers running RIP send and receive reachability information every 30 seconds on UDP port 520. An update message is sent to all the router's neighbors whenever an update from another router causes changes to its forwarding table. RIP is actually a straightforward implementation of distance-vector routing with a link cost of 1. It always finds the route with the minimum number of hops. RIP does not take the link speed or traffic level into consideration. Valid distances are from 1 to 15, with 16 representing infinity or unreachable. Thus RIP is limited to running on fairly small networks (i.e. networks that do not have paths longer than 15 hops). With the introduction and use of

subnets and Classless Inter-Domain Routing (CIDR), RIPv2 is developed to support it, since RIPv1 cannot be used with variable length subnetting.

RIPv1 is a simple distance vector protocol. It has been enhanced with various techniques, including Split Horizon and Poison Reverse in order to enable it to perform better in somewhat complicated networks. However, being a simple distance vector protocol, it will run into difficulty. First and foremost, it will occasionally have to count to infinity in order to purge bad routes. This delays the convergence of routing. To ensure quick convergence, RIPv1 defines infinity as 16 hops. That means that networks with diameters larger than 16 cannot use RIPv1. Even getting close to that limit can cause confusion for some implementations. The way to overcome this problem to use RIPv2. RFC 1723 recommends that RIPv1 be used only in networks with simple topologies and simple reachability.

RIPv1 includes no security functions while RIPv2 includes a mechanism for authenticating the sender of the routing information. Sites which are worried about the vulnerability of their routing infrastructure and which feel they must run a RIP-like protocol should use RIPv2.

B. OPEN SHORTEST PATH FIRST (OSPF)

OSPF is a link-state routing protocol. As such, it calls for the sending of *Link State Advertisements* (LSAs) to all other routers within the same hierarchical area of a domain. In OSPF, LSAs are refreshed at a minimum of every 30 minutes. New advertisements are sent out more frequently when some part of the topology changes. As OSPF routers accumulate link state information, they use *the Shortest Path First* (SPF) algorithm to calculate the shortest path to each node. OSPF is designed to run internal to a single Autonomous System; hence it is classified as an Interior Gateway Protocol (IGP). The OSPF protocol has been designed for the TCP/IP Internet environment, including explicit support for CIDR and the tagging of externally-derived routing information.

As a link state routing protocol, OSPF contrasts with RIP, which is a distance vector routing protocol. Routers running the distance vector algorithm send all or a portion of their routing tables in routing update messages, but only to their neighbors.

Link state algorithm flood routing information to all nodes in an AS. However, each router sends only that portion of the routing table that describes the state of its own links.

C. BORDER GATEWAY PROTOCOL

The first Interdomain Routing Protocol was the Exterior Gateway Protocol (EGP). EGP has many limitations and forces a tree-like topology onto the Internet. Hence EGP was replaced by the Border Gateway Protocol (BGP) which treats the Internet as an arbitrarily interconnected set of ASs. There are several versions of BGP and the current version is 4.

The Border Gateway Protocol (BGP), defined in RFC 1771, enables loop-free interdomain routing between Autonomous Systems (AS). Its purpose is to exchange reachability information with other ASs. The concept of reachability is analogous to a statement that "the network could be reached through this AS." Each AS may run its own intradomain routing protocol. They may have different routing metrics and thus impossible to calculate meaningful path costs for a path that crosses multiple ASs. Routers in an AS can also use multiple interior gateway protocols to exchange routing information inside the AS and an exterior gateway protocol to route packets outside the AS. Routers that belong to the same AS and running BGP to exchange reachability information are said to be running Internal BGP (IBGP). Routers that belong to different AS and running BGP are said to be running External BGP (EBGP).

To appreciate the significance of BGP, let us assume that you are administering an AS and that it is connected to the Internet without running BGP to its provider. A default route towards the provider will have to be created. All non-local packets will go out via the interface specified by the router. Its provider will probably put static routes towards it, and redistributes those static routes into their IGP, and consequently into BGP that's probably connected to another AS upstream. Under this situation, if you have any address space "inside" of your provider's larger "netblock" or "aggregate", you won't be able to advertise it to the outside world specifically – your provider will only advertise their larger block. If you have other networks, your provider will just statically announce those routes to the world.

However, if you have BGP running with your provider, you will be able to gather all the routes your providers have while they can listen to your route announcements and then redistribute some or all of those to their neighbors and customers. The net difference is that they can now advertise a more specific route for you. This is important as the primary rule of IP routing is "The most specific route always wins".

APPENDIX B – SAAM SERVER.SERVER CLASS CODE

```
//23Feb2000[Henry]     - modified FlowRequest and FlowResponse,
//                       added SLSDbase, etc
// Feb 2000 [akkoc]    - modified
//01august99[vrable]   - created

package saam.server;

import saam.EmulationTable;
import saam.Translator;
import saam.*;

import saam.net.*;
import saam.message.*;
import saam.control.*;
import saam.event.*;
import saam.router.*;
import saam.util.*;
import java.net.*;
import java.util.*;
import java.io.*;

import saam.server.diffserv.*;


/**
 * The <em>Server</em> is an object within the SAAM architecture that
 * maintains a picture of the network for use in assigning flows to
paths.
 */

public class Server implements  Runnable{

  //declare class variables

  /** Contains what is known about the network. */
  //private PathInformationBase PIB;
  private ClassObjectStructure PIB;

  /** Enables the Server to receive and send particular types of
messages. */
  private ControlExecutive controlExec;

  /** A maximum number of hops that a search for different paths may
take. */
  private int Hmax = 4;

  /**
   * Used to lookup what flow id should be used to send out control
messages
   * to specified routers.
   */
  private Hashtable flowLookUp = new Hashtable();
```

```
  /** Used to assign the right number of service level pipes to
interfaces in
   * this SAAM region. Only used during initialization -- later were
assume
   * SLPs are known to routers
   */
  private int numOfServiceLevels = 5;//4;

  public static final byte NUMBEROFSERVICELEVELS = 5;
  public static final byte CONTROL_SERVICELEVEL = 0;
  public static final byte IS_SERVICELEVEL = 1;
  public static final byte DS_SERVICELEVEL = 2;
  public static final byte BE_SERVICELEVEL = 3;
  public static final byte OTHER_SERVICELEVEL = 4;  //e.g. tagged
packets

  public static float[] throughputRatioForSL = new
float[NUMBEROFSERVICELEVELS];

  /** The database that stores all SLS allocated in the network */
  private SLSDbase slsDbase;
  /** A reusable SLSTable for temperary storage of all SLS of a node */
  private SLSTable slsTable;
  /** The vector containing all flow request and response result */
  private Vector flowTableData = new Vector();

  private int IS_RejectionRate = 0;
  private int DS_RejectionRate = 0;
  public static int IS_REJECTIONTHRESHOLD = 3;
  public static int DS_REJECTIONTHRESHOLD = 4;

  private FileIO logfile;  //added by Henry
  private Date date = new Date();  //added by Henry (used for logfile)

  /**
   * The value assigned to flow ids that can not be supported. This
should be
   * switched over to 0 as soon as routers are converted.
   */
  //added by Henry
  public static int FLOWUNSUPPORTABLE = 0;
  public static int FLOWNUNREACHEABLE = -1;
  public static int FLOWNEGOTIABLE = -2;
//  public static float INCREMENTALFACTOR = 1.1f;
//  public static float NEGOTIABLEFACTOR = 0.75f;
//  public static float BORROWABLEFACTOR = 0.1f;
//  public static float DS_LOWUTILITYFACTOR = 0.5f;
  public static float DS_LENDINGFACTOR = 0.05f;
  public static int FLOWNOTSUPPORTABLE = 99;   //no longer used

  /*************************************************************/

  public static int INITIALDELAY = 0;
  public static int INITIALLOSSRATE = 0;
  public static int INITIALTHROUGHPUT = 10000;

  public static int RETURNFLOWDELAY = 50;
```

```
public static int RETURNFLOWLOSSRATE = 50;
public static int RETURNFLOWTHROUGHPUT = 1000;

public static int ROUTERNOTINPIB = 0;

public static int NOSUPPORTABLEPATHINPIB = 0;

public static int SERVERNODEID = 1;

public static int FLOWTOSERVER = 0;

public static int PSUEDORANDOMSOURCEPORT = 8000;

public static int INITIALPATHID = 0;

public static int INITIALHEIGHTOFSEARCH = 1;
public static int INCREMENTATIONOFSEARCH = 1;
public static int DESTINATIONNODE = 0;

public static int INITIALZERO = 0;


/** Defines with the appropriate IPv6 address of this server. */
//private String serverIPv6 = controlExec.getServerIP().toString();
// private String serverIPv6 = "99.99.99.0.0.0.0.0.0.0.0.0.0.0.0.1";

/** Time when the all possible paths were found. */
private long timeOfLastPIBBuild = System.currentTimeMillis();

/**
 * The amount of time that we want to have between rebuilding of
paths. This
 * is not currently implemented.
 */
private long timeBetweenPIBBuilds = 120000; // 2 minutes (or 120 sec)

/** A boolean that will allow the showing of comments. */
private boolean showComments = true;

private SAAMRouterGui gui;

/**
 * added by hasan uysal to process lsa for the
 */
private Hashtable IPv6ToIntIdTable=new Hashtable();


// 2000 akkoc added
private int sequenceNumber = 1;
private static final int CTS = 0;    //SINCE ITS SERVER BY ITSELF
private int hopCount;


private static byte serverType;
private static int flowId;
private static byte metricType;
private static int cycleTime;
```

```
  private static int globalTime;

  //1 Feb 2000 akkoc added
  private IPv6Address ServerId;

  boolean cofMesOK=false;
  Object theLock = new Object();
  Thread configThread;

  /**
   * Constructs a server that will use a specified type of <em>Path
Information
   * Base</em>.  The PIB may be in the form of a database structure
(which
   * requires an existing ODBC configured local database) or a class
   * object structure. The control executive is the interface to the
IPv6
   * protocol stack, in order for messages to flow to and from the
network.
   * The final step taken is the deletion of all existing data, which
is
   * important only in a database structure since a class object
structure is
   * volatile.
   * @param type The type of structure that the PIB is to assume.
   * @param controlExec The control executive that will exchange
messages
   * with this server.
   */
  public Server(String type, ControlExecutive controlExec){
    //if (type == "database")
      //PIB = new DatabaseStructure();
    //else
      PIB = new ClassObjectStructure();

    this.controlExec = controlExec;
    gui=new SAAMRouterGui("Server");

    // 1feb 2000 akkoc added
    ServerId = controlExec.getRouterId();

    PIB.deleteAllData();

    initResourceAllocation();
    this.slsDbase = new SLSDbase();
    setupSLSDbase();
    logfile = new FileIO();
    logfile.openToWrite("server\\log.dat");
  }


  //Added by Henry for testing only
  public Server(SAAMRouterGui gui, ControlExecutive controlExec,
              ClassObjectStructure PIB, SLSDbase slsDbase){
    this.gui = gui;
    this.controlExec = controlExec;
    this.PIB = PIB;
```

```
    PIB.deleteAllData();

    initResourceAllocation();
    this.slsDbase = slsDbase;
    //setupSLSDbase();
    logfile = new FileIO();
    logfile.openToWrite("server\\log.dat");
    showComments = false;
  }


  /**
   * Assigns respective bandwidth allocation for various service levels
   */
  private void initResourceAllocation(){
  //initialize amount of trunk allocation
    throughputRatioForSL[CONTROL_SERVICELEVEL] = 0.1f;//Control Packets
    throughputRatioForSL[IS_SERVICELEVEL] = 0.2f;     //INTSERV
    throughputRatioForSL[DS_SERVICELEVEL] = 0.2f;     //DIFFSERV
    throughputRatioForSL[BE_SERVICELEVEL] = 0.2f;     //BESTEFFORT
    throughputRatioForSL[OTHER_SERVICELEVEL] = 0f;    //Tag Packets
  }


//*************************************************************************
*****
// These methods handle external network communications from routers
//*************************************************************************
*****/


/*************************************************************************
*****/

//Hasan UYSAL

 /**
   * Receives link state advertisement messages from router and
processes the
   * service level pipe status information that they contain. It begins
by
   * checking to see if a router with the interface address described
by this
   * LSA is known to the PIB. If such a router is known to exist, it
then
   * checks to see if the service level pipe described by this LSA is
known to
   * the PIB. If the service level pipe is known, then update its
status.
   * Otherwise, add the SLP with the specified QoS characteristics.
Finally,
   * update the effective QoS for the paths that pass over this service
level
   * pipe by calling the determineEffectiveQoSForPaths().
   * @param router A representation of a router as defined by an LSA.
   */
  public synchronized void processLSA(LinkStateAdvertisement LSA) {
```

```
    System.out.println("Started ProcessLSA");
    //who is the generating router
    IPv6Address routerId = LSA.getMyIPv6();
    gui.sendText("An LSA arrived at "+System.currentTimeMillis()+" from
"+routerId.toString());
    long startTs,endTs;

    Vector IntLSAs = LSA.getLSAs();
    Vector ips = new Vector();

    boolean newRouter = false;
    int nodeId = ROUTERNOTINPIB;
    String keyStr = routerId.toString();

    startTs=System.currentTimeMillis();

    //check if IPv6ToIntIdTable contains this router
    if(!IPv6ToIntIdTable.containsKey(keyStr)){
      System.out.println("This router is not in my table. \nTaking
InterfaceLSAs from LSA message.");
      Enumeration enum = IntLSAs.elements();
      while(enum.hasMoreElements()){
        InterfaceLSA tempLsa=(InterfaceLSA)enum.nextElement();
        IPv6Address tempIp=tempLsa.getIP();
        ips.add(tempIp);
      }
      //check if there is a router with these interafces
      //this means we removed an interface which was a router id
earlier and
      //routerid has changed in the table
      nodeId=PIB.doesRouterExist(ips);

      //if the router is not in the pib it is a new one
      if(nodeId==this.ROUTERNOTINPIB){
        System.out.println("Router is a new router.");
        newRouter = true;
        nodeId=PIB.getNewNodeId();
        this.IPv6ToIntIdTable.put(keyStr,new Integer(nodeId));
      }
      else{ //it is not a new one this lsa is a second copy of the
removal LSA
        return;
      }
    }//if

    if(newRouter){
      gui.sendText("this is a new router and will be aded to the
PIB.");
      //addd all interfaces to the PIB and compute the Paths
      int IdInt = ((Integer)IPv6ToIntIdTable.get(keyStr)).intValue();
      InterfaceLSA newInterface=null;
      Enumeration enum=IntLSAs.elements();
      while(enum.hasMoreElements()){
        newInterface=(InterfaceLSA)enum.nextElement();
        this.checkAndAdd(IdInt,newInterface);
      }
```

```
      findAllPossiblePaths();
      determineEffectiveQoSForPaths();
      //Added by Henry
      System.out.println("initializing Resources of router "+routerId);
      initializeResourceAllocation(routerId);
      if (!routerId.equals(ServerId)){
       System.out.println("sending SLSTable to router...");
       sendSLSTable(routerId, new Integer(nodeId));
      }
      //*****************************************************
      return;
    }

    System.out.println("in ProcessLSA 2");

    //it may be a new or an old router take the int id of the router
    nodeId = ((Integer)IPv6ToIntIdTable.get(keyStr)).intValue();

    Enumeration lsaInterfaceEnum = IntLSAs.elements();
    while(lsaInterfaceEnum.hasMoreElements()){
      InterfaceLSA
curInterface=(InterfaceLSA)lsaInterfaceEnum.nextElement();
      byte type = curInterface.getLSAType();
      gui.sendText("Type of the InterfaceLSA is "+(int)type);
      switch(type){
        case InterfaceLSA.ADD:
          checkRouterId(routerId,IntLSAs);
          checkAndAdd(nodeId,curInterface);
          if(!newRouter){ //another interface is added to the router
                          //PIB will be updated
            findAllPossiblePaths();
            determineEffectiveQoSForPaths();
          }
        break;

        case InterfaceLSA.UPDATE:
          updatePIB(nodeId,curInterface);
        break;

        case InterfaceLSA.REMOVE:
          checkRouterId(routerId,IntLSAs);
          removeInterfaceFromPIB(nodeId,curInterface);
        break;

        default:
          gui.sendText("Interface LSA type is not a recognized type.");
      }//end switch
    }//end while

    System.out.println("end of ProcessLSA");
  } //end processLSA



  private void checkRouterId(IPv6Address routerId,Vector iLsaVector){
    IPv6Address tempIP;
    IPv6Address tempId=new IPv6Address();
```

```
    InterfaceLSA tempIntLsa;
    byte[] idBytes;
    byte[] tempBytes;
    Enumeration enum=iLsaVector.elements();
    while(enum.hasMoreElements()){
      tempIntLsa = (InterfaceLSA)enum.nextElement();
      if(tempIntLsa.getLSAType()==InterfaceLSA.REMOVE){
        continue;
      }
      tempIP=tempIntLsa.getIP();
      idBytes=tempId.getAddress();
      tempBytes= tempIP.getAddress();
      for(int i=0;i<IPv6Address.length;i++){
        if(idBytes[i]>tempBytes[i]){
          break;
        }
        if(idBytes[i]<tempBytes[i]){
          tempId=tempIP;
          break;
        }
      }
    }

    if(tempId.equals(routerId)){//there is no change in id
      return;
    }
    //there is change in the router id
    //old router id has to be changed from the table
    int
knownId=((Integer)IPv6ToIntIdTable.get(routerId.toString())).intValue()
;
    IPv6ToIntIdTable.remove(routerId.toString());
    routerId=tempId;
    IPv6ToIntIdTable.put(tempId.toString(),new Integer(knownId));
  }//end checkRouterId()



  private void checkAndAdd(int nodeId,InterfaceLSA curInterface){

    IPv6Address ip=curInterface.getIP();
    int bandwidth=this.INITIALZERO;
    //Is this interface in my Path Information Base

      bandwidth = curInterface.getBandwith();

      if(!PIB.doesLinkExist(ip)){
        PIB.addLink(ip,bandwidth);
      }//end inner if

      //now add interface
      PIB.addInterface(nodeId,ip);

      //added by Henry
      for (int service_level = 0;
          service_level < NUMBEROFSERVICELEVELS; service_level++){
          PIB.addSLP(ip, service_level, INITIALDELAY, INITIALLOSSRATE,
```

```
        //     INITIALTHROUGHPUT);

(int)(throughputRatioForSL[service_level]*INITIALTHROUGHPUT));
      }

  }

  private void updatePIB(int nodeId,InterfaceLSA iLsa){
    byte slps=iLsa.getNumOfSLPs();
    Vector slpVector=iLsa.getSLPs();
    IPv6Address ip=iLsa.getIP();
    for(int i=0;i<slps;i++){
      SLPLSA slpLsa=(SLPLSA)slpVector.elementAt(i);
      byte slpNumber=slpLsa.getSLPNum();
      byte utilization=slpLsa.getUtilization();
      short delay = slpLsa.getDelay();
      short lossRate = slpLsa.getLossRate();

PIB.updateSLP(ip,slpNumber,delay,(int)(lossRate/100),(utilization/2));
    }
  }


  private void removeInterfaceFromPIB(int nodeId,InterfaceLSA
curInterface){
     gui.sendText("Removing interface from PIB.");
     removePathsTraversingInterface(curInterface);
     removeLinkFromPIB(curInterface.getIP());
     removeInterfaceFromNode(curInterface.getIP());
  }

  private void removePathsTraversingInterface(InterfaceLSA iLsa){
    gui.sendText("Removing Paths using the interface from PIB.");
    for(int i=0;i<iLsa.getNumOfSLPs();i++){
      Vector pathIds=PIB.getAllPathIdsThatTraverseSLP(iLsa.getIP(),i);

      ClassObjectStructure cos=(ClassObjectStructure)PIB;
      cos.deletePathsTraversingInterface(pathIds);

    }
  }

  private void removeLinkFromPIB(IPv6Address ip){
    gui.sendText("Removing linkof the interface from PIB.");
    IPv6Address netIP=ip.getNetworkAddress();
    ClassObjectStructure cos=(ClassObjectStructure)PIB;
    cos.links.remove(netIP.toString());
  }

  private void removeInterfaceFromNode(IPv6Address ip){
    gui.sendText("Removing Interface from nodes.");
    ClassObjectStructure cos=(ClassObjectStructure)PIB;
    cos.nodes.remove(ip.toString());
  }

/*********************************************************************
*****/
```

```
  /**Used only by Henry to build up PIB for local test
   * Receives Hello messages from routers and then processes them.  It
starts
   * building a vector of IPv6Addresses from the interfaces included in
the
   * Hello message. This vector is passed to the PIB's
doesRouterExist() which
   * determines if a router with any of these interfaces have been
identified
   * before. If this is a new router, a new unique node id is
assigned.<p>
   * For each of the interfaces identified in the Hello message, if
this
   * interface was is not known to the PIB, check to see if the
corresponding
   * link is known to the PIB. If this link is not known to the PIB,
add it.
   * Next, add the new interface between the node and link. Also, add
each
   * service level pipe that is assigned within this SAAM region.<p>
   * The next step is to rebuild the paths that are possible across the
network
   * now considering this new hello message. The frequency of these
rebuilds is
   * not a major concern in a controlled environment, but will need to
be
   * addressed later. Finally, a flow request is create and received
for
   * communicating back to this node. This is only possible if the
PIB's
   * determineAllPossiblePaths() has been executed after the processing
of this
   * particular hello message, if this a new router. After all paths to
each
   * known router are found, we finish this method with a call to
   * determineEffectiveQoSForPaths(). The call to
   * determineEffectiveQoSForPaths() ensures that even if no QoS
parameters are
   * known about these new parts of the network, that at least some
initial
   * values will be assigned. This initialization allows the new paths
to be
   * assigned if needed.
   * @param hello An initialization message from a router.
   */
  public void processHello(Hello hello) {

    long start, finish;
    Vector interfaces;
    int node_id = INITIALZERO;
    InterfaceID myInterface;
    int bandwidth = INITIALZERO;
    IPv6Address address = new IPv6Address();
    Vector IPv6Addresses = new Vector();
    boolean newRouter = true;
    FlowRequest myFlowRequest = new FlowRequest();
```

```
    // capture the start time of processing a hello
    start = System.currentTimeMillis();

    // produce a vector of IPv6Addresses
    interfaces = hello.getInterfaceIDs();
    for (int i = INITIALZERO; i < interfaces.size(); i++){
      address = ((InterfaceID)interfaces.elementAt(i)).getIPv6();
      IPv6Addresses.addElement(address);
    }

    // check if router exists and if so, return it's node id, else
return 0
    node_id = PIB.doesRouterExist(IPv6Addresses);

    // if the router does not exist in PIB
    if (node_id == ROUTERNOTINPIB){
      // assign it a new node id
      node_id = PIB.getNewNodeId();
    } else {
      newRouter = false;
    }

    // run through all of the LSA interfaces
    for (int i = INITIALZERO; i < interfaces.size(); i++) {
      myInterface = (InterfaceID)interfaces.elementAt(i);
      address = myInterface.getIPv6();
      // if a new interface is not found in the PIB, then ...
      if (!PIB.doesInterfaceExist(address)){
        bandwidth = myInterface.getBandwidth();
        address = myInterface.getIPv6();
        // if the link is not contained in the PIB, then add it
        if (!PIB.doesLinkExist(address)){
          PIB.addLink(address, bandwidth);
        }
        // now add the interface between the node and the link
        PIB.addInterface(node_id, address);
        // now add each service level pipe
        for (int service_level = 0;
          service_level < NUMBEROFSERVICELEVELS; service_level++){
          PIB.addSLP(address, service_level, INITIALDELAY,
INITIALLOSSRATE,
            //INITIALTHROUGHPUT);

(int)(throughputRatioForSL[service_level]*INITIALTHROUGHPUT));
        }
      } // end if
    } //end interfaces for

    // capture the hello processing finish time
    finish = System.currentTimeMillis();
      gui.sendText("Server: processHello: Time required = "
        +(finish-start)+" milliseconds.");

    //time since last PIB build is > 2 min and if node did not exist
before
```

```
    //if ((timeOfLastPIBBuild - System.currentTimeMillis())
>timeBetweenPIBBuilds
    //                                                          &&
newRouter){

      // rebuild all possible paths
      findAllPossiblePaths();

      // determine effective QoS of each path
      determineEffectiveQoSForPaths();

      // construct a new flow to this router
      /*try{
        myFlowRequest = new
FlowRequest(IPv6Address.getByName(serverIPv6),
          address,System.currentTimeMillis(),RETURNFLOWDELAY,
          RETURNFLOWLOSSRATE,RETURNFLOWTHROUGHPUT);
      } catch(UnknownHostException uhe){
        System.err.println("Server: main: UnknownHostException: " +
uhe);
      }
      processFlowRequest(myFlowRequest);        */
    //}

  } //end processHello


  /**
   * Receives and processes flow requests from applications. It begins
   * by finding a source and a destination router. These routers may be
where
   * the applications are residing themselves, which is our standard
situation.
   * The application could, however, reside on some host that is not
registered
   * with the PIB as a router. In this case, the appropriate source or
   * destination router would be a router connected to the same link.
<p>
   * The PIB is checked to ensure that there is the effective QoS
available on
   * some path to satisfy the request. If a satisfactory path is found,
a new
   * unique flow id is assigned and this new flow is associated with
that path.
   * Each router in the path is retrieved and a new flow routing table
entry is
   * sent to each. If no path can provide the requested level of QoS,
then the
   * flow is assigned to zero, which will be interpreted by IPv6 as
best effort
   * traffic. Finally, a flow response is sent back to the application
to
   * inform it of its assigned flow id. If the flow id that is return
is zero,
   * it will be the application's responsibility to either lower it QoS
request
   * or to send its traffic as best effort.
```

```
   * @param flow_request The message requesting the establishment of a
flow.
   */
  public void processFlowRequest(FlowRequest flow_request) {

    int source_router, destination_router, path_id,
      flow_id=FLOWNUNREACHEABLE;
    long start, finish;
    byte service_Type;

    // capture the start time of processing a flow request
    start = System.currentTimeMillis();

    // find a router on the same subnet as the source host
    source_router =
      PIB.findARouterOnLink(flow_request.getSourceInterface());

    // find a router on the same subnet as the destination host
    destination_router =
      PIB.findARouterOnLink(flow_request.getDestinationInterface());

    //added by Henry
    service_Type = flow_request.getServiceLevel(); //a service request?

    if (showComments){
      gui.sendText("Server: processFlowRequest: from node "
          +source_router+"for service level "+service_Type);
    }

    if (service_Type == IS_SERVICELEVEL) {
      IS_Admission(source_router, destination_router, flow_request);
    }
    else if (service_Type == DS_SERVICELEVEL) {
      DS_Admission(source_router, destination_router, flow_request);
    }


/*****************************************************************/

    //else if (service_Type == CONTROL_SERVICELEVEL) {
    else {  //for backward compatibility, no reason for other service
level
      path_id = PIB.getPathThatCanSupportFlowRequest(source_router,
                      destination_router, flow_request);
      // if a path can support this request, then...
      if(path_id != NOSUPPORTABLEPATHINPIB){
        // assign a flow id to the request
        flow_id =
PIB.getNewFlowId(path_id,source_router,destination_router,
                                                    flow_request);
        updateRouter(source_router, destination_router, path_id,
flow_id);
      }// end if
    }// end if
//Old code follows
    //give routers time to finish updating tables
    try{
```

```
      Thread.sleep(2000);
    }catch(InterruptedException ie){
      gui.sendText(ie.toString());
    }
     // if the source of this flow is the server,
    if (source_router == SERVERNODEID) {
      // then add this new flow to hash table for later lookup
      if (showComments){
        gui.sendText("Server: processFlowRequest: use flow "+flow_id
          +" to send to node "+destination_router);
      }
      if (destination_router == SERVERNODEID) {
        flow_id = FLOWTOSERVER;
      }
      flowLookUp.put(new Integer(destination_router),new
Integer(flow_id));
      sendFlowResponse(flow_request, flow_id);
    }

    // capture the flow request processing finish time
    finish = System.currentTimeMillis();
    gui.sendText("Server: processFlowRequest: Time required = "
      +(finish-start)+" milliseconds.");
  }


  /**
   * Determines and updates the flow routing table in all the routers
   * affected by the flow.
   * @param source_router The router requesting for the flow.
   * @param destination_router The destination of a flow.
   * @param path_id The path of a flow
   */
  private void updateRouter(int source_router, int destination_router,
                            int path_id, int flow_id) {
    Vector slps_in_path;
    SLPSequence currentSLPSequence,nextSLPSequence = new SLPSequence();
    int SLP_source_router, SLP_destination_router, service_level;
    IPv6Address link_id = new IPv6Address();
    IPv6Address next_hop;
    IPv6Address sourceAddress;

    // determine each router in path
    // transmit Flow Routing Table Entry to it
    slps_in_path = PIB.getSLPSequenceOfPath(path_id);
    // for each router in the path, send a FRTE update
    for (int index = INITIALZERO; index < slps_in_path.size();
index++){
      // assign new slp sequence object
      currentSLPSequence = (SLPSequence)slps_in_path.elementAt(index);

      // if not the last link..
      if (index+1 != slps_in_path.size()){
        nextSLPSequence = (SLPSequence)slps_in_path.elementAt(index+1);
      }

      // retrieve values from this object
```

```
      SLP_source_router = currentSLPSequence.getSourceRouter();
      link_id = currentSLPSequence.getLinkId();
      service_level = currentSLPSequence.getServiceLevel();

      // if not the last link...
      if (index+1 != slps_in_path.size()){
        SLP_destination_router = nextSLPSequence.getSourceRouter();
      } else {
        // else it is the destination node of the flow
        SLP_destination_router = destination_router;
      }
      // determine destination address for next hop
      next_hop = PIB.getInterfaceAddress(SLP_destination_router,
link_id);

      // determine source address
      sourceAddress = PIB.getInterfaceAddress(SLP_source_router,
link_id);

      // send the flow routing table entry update
      sendFRTEUpdate(sourceAddress, flow_id, next_hop, service_level);

    } // end for

    //give routers time to finish updating tables
    try{
      Thread.sleep(2000);
    }catch(InterruptedException ie){
      gui.sendText(ie.toString());
    }

  }//end updateRouter


  /**
   * Receives flow termination from routers and then processes them.
   */
  //public void receiveFlowTermination() { }

  /**Henry
   * Receives flow termination from routers and then processes them.
   * synchronized for testing
   */
  public synchronized void receiveFlowTermination(int flow_id) {
    gui.sendText("receiveFlowTermination for flow_id: "+flow_id);
    PIB.deleteAssignedFlow(flow_id);
  }

  /**Henry
   * Receives a SLSTableEntry message from a router that contains
   * the SLS that is to be removed from a SLSTable in the SLSDbase.
   */
  public void receiveSLSTableUpdate(SLSTableEntry message) {
    int user_id = message.getUserId();
    int node_id = message.getNodeId();
    gui.sendText("receivedSLSTableUpdate for user: "+user_id
            +" at Node "+node_id);
```

```
    deleteSLS(new Integer(node_id), user_id);
  }

  /**Henry
   * Used for local test to remove a SLS from a SLSTable in the
SLSDbase.
   */
  public synchronized void receiveSLSTableUpdate(int user_id) {
    Vector allNodeId = PIB.getAllRouterIds();
    gui.sendText("receivedSLSTableUpdate for user: "+user_id);
    //deleteSLS(allNodeId, user_id);
    for (int i=0; i<allNodeId.size(); i++ ) {
      Integer node_id = (Integer)allNodeId.elementAt(i);
      deleteSLS(node_id, user_id);
    }
  }

  /**Henry
   * Admission control for Integrated Service flows
   * @param   source  The node id of the source router
   * @param   destination  The node id of the destination router
   * @param   flow_request  The flow request message
   * @return  The flow response that contain the result of the
   * admission control.
   */
  private synchronized FlowResponse IS_Admission(int source,
    int destination, FlowRequest flow_request) {

    int flow_id = FLOWNUNREACHEABLE;
    int supporting_path = 0;
    int used_throughput = 0;
    byte result = FlowResponse.UNREACHEABLE;
    FlowResponse response;
    Vector slps;
    IPv6Address[] pathAddress;
    SLP nextSLP;
    int path_id = 0;

    gui.sendText("\nProcessing IS Admission Control...");
    int throughput = flow_request.getRequestedThroughput();
    IPv6Address sourceAddress = flow_request.getSourceInterface();
    //find a physical path that can reach the destination requested
    supporting_path = PIB.getPathThatSupportFlowRequest(source,
                      destination, flow_request);
    gui.sendText("Requested throughput = " + throughput);
    //if destination is unreacheable
    if (supporting_path == FLOWNUNREACHEABLE) {
      gui.sendText("UNREACHEABLE!");
      sendFlowResponse(flow_request, flow_id,
FlowResponse.UNREACHEABLE);
      result = FlowResponse.UNREACHEABLE;
      response = new FlowResponse(flow_request.getTimeStamp(),
                 result, flow_id);
    }
    else if (supporting_path == FLOWUNSUPPORTABLE) {
      //destination is reacheable but network cannot meet
      //log down event
```

```
      gui.sendText("FLOWUNSUPPORTABLE..... rejected.");
      sendFlowResponse(flow_request, flow_id, FlowResponse.REJECTED);
      result = FlowResponse.REJECTED;
      response = new FlowResponse(result, flow_id);
      gui.sendText("Request cannot be met event logged.");
      IS_RejectionRate++;
      gui.sendText("IS_RejectionRate: "+IS_RejectionRate);
      if (IS_RejectionRate > IS_REJECTIONTHRESHOLD) {
        gui.sendText("IS_REJECTIONTHRESHOLD exceeded");
      }
      logfile.write(""+date.toString()+"
"+flow_request.toString()+"\n");
    }
    else {
      gui.sendText("Path that support flow request = "
            /*+ flow_request*/ + supporting_path);
      //get all interface addresses of the path
      pathAddress = PIB.getPathAddress(supporting_path);
      // assign a flow id to the request
      flow_id = PIB.getANewFlowId(supporting_path, source,
                  destination, flow_request);
      //update PIB (observed values will be updated by LSA update
      for (int i=0; i<pathAddress.length; i++) {
        IPv6Address address = pathAddress[i];
        gui.sendText("RemainingThroughput (before) for Interface: "
            +pathAddress[i]+" = "+PIB.getRemainingThroughput(
            //pathAddress[i], Server.IS_SERVICELEVEL));
            address, Server.IS_SERVICELEVEL));
        PIB.updateRemainingBWOfAllPaths(pathAddress[i],
            Server.IS_SERVICELEVEL, throughput);
        PIB.updateSLP(pathAddress[i], Server.IS_SERVICELEVEL,
            flow_request.getRequestedDelay(),
            flow_request.getRequestedLossRate(), throughput);
        gui.sendText("AllocatedThroughput for Interface: "
            +pathAddress[i]+" = "+throughput);
        gui.sendText("RemainingThroughput (after) for Interface: "
            +pathAddress[i]+" = "+PIB.getRemainingThroughput(
            //pathAddress[i], Server.IS_SERVICELEVEL));
            address, Server.IS_SERVICELEVEL));
      }
      //update flowTable of all routers in the path
      updateRouter(source, destination, supporting_path, flow_id);
      sendFlowResponse(flow_request, flow_id,
FlowResponse.IS_ACCEPTED);
      result = FlowResponse.IS_ACCEPTED;
      response = new FlowResponse(flow_request.getTimeStamp(), result,
flow_id);
    }
    return response;
  }

  /**Henry
   * Admission control for Integrated Service flows
   * @param   source  The node id of the source router
   * @param   destination  The node id of the destination router
   * @param   flow_request  The flow request message
   * @return  The flow response that contain the result of the
```

```
   * admission control.
       */
  public synchronized FlowResponse IS_Admission(
                    FlowRequest flow_request) {

    FlowResponse response;
    int source =
      PIB.findARouterOnLink(flow_request.getSourceInterface());

    // find a router on the same subnet as the destination host
    int destination =
      PIB.findARouterOnLink(flow_request.getDestinationInterface());

    response = IS_Admission(source, destination, flow_request);
    return response;
  }

  /**
   * Admission control for Differentiated Service flows
   * @param   source  The node id of the source router
   * @param   destination  The node id of the destination router
   * @param   flow_request  The flow request message
   * @return  The flow response that contain the result of the
   * admission control.
       */
  private synchronized FlowResponse DS_Admission(
      int source, int destination, FlowRequest flow_request) {
    int[] supportable_paths;
    int supporting_path = 0;
    int used_throughput = 0;
    Vector slps;
            FlowResponse response;
            IPv6Address[] pathAddress;
    SLP nextSLP;
    int path_id = 0;

    gui.sendText("\nProcessing DS Admission Control...");
    int throughput = flow_request.getRequestedThroughput();
    IPv6Address sourceAddress = flow_request.getSourceInterface();
    //find a physical path that can reach the destination requested
    supporting_path = PIB.getPathThatSupportFlowRequest(source,
                      destination, flow_request);
    gui.sendText("Path that support flow request: "
      + flow_request + " is " + supporting_path);
    if (supporting_path == FLOWNUNREACHEABLE) {  //if destination is
unreacheable
      gui.sendText("UNREACHEABLE!");
      sendFlowResponse(flow_request, FlowResponse.UNREACHEABLE, null);
      response = new FlowResponse(flow_request.getTimeStamp(),
            FlowResponse.UNREACHEABLE);
    }
    else if (supporting_path == FLOWUNSUPPORTABLE) {
      //log down event
      sendFlowResponse(flow_request, FlowResponse.SLA_NOT_AVAILABLE,
null);
        response = new FlowResponse(flow_request.getTimeStamp(),
                                    FlowResponse.SLA_NOT_AVAILABLE);
```

```
      gui.sendText("Request cannot be met event logged.");
      DS_RejectionRate++;
      gui.sendText("DS_RejectionRate: "+DS_RejectionRate);
      if (DS_RejectionRate > DS_REJECTIONTHRESHOLD) {
        gui.sendText("DS_REJECTIONTHRESHOLD exceeded");
      }
      logfile.write(""+date.toString()+"
"+flow_request.toString()+"\n");
    }
    else {
      //get all interface addresses of the path
      pathAddress = PIB.getPathAddress(supporting_path);
      //update PIB (should be done by LSA update
      for (int i=0; i<pathAddress.length; i++) {
        gui.sendText("RemainingThroughput (before) for Interface: "
            +pathAddress[i]+" = "+PIB.getRemainingThroughput(
            pathAddress[i], Server.DS_SERVICELEVEL));
        PIB.updateRemainingBWOfAllPaths(pathAddress[i],
            Server.DS_SERVICELEVEL, throughput);
        PIB.updateSLP(pathAddress[i], Server.DS_SERVICELEVEL,
            flow_request.getRequestedDelay(),
            flow_request.getRequestedLossRate(), throughput);
        gui.sendText("AllocatedThroughput for Interface: "
            +pathAddress[i]+" = "+throughput);
        gui.sendText("RemainingThroughput (after) for Interface: "
            +pathAddress[i]+" = "+PIB.getRemainingThroughput(
            pathAddress[i], Server.DS_SERVICELEVEL));
      }
      //create a new ServiceLevelSpec and add it to the SLSDbase
      SLS newSLS = addSLS(new Integer(source), flow_request);
      slsDbase.displaySLSTable();  //display SLSDbase
      controlExec.updateSLSTable();  //only needed for displaying
SLSTable
      sendFlowResponse(flow_request, FlowResponse.DS_ACCEPTED, newSLS);
      response = new FlowResponse(flow_request.getTimeStamp(),
          FlowResponse.DS_ACCEPTED, flow_request.getUser(), newSLS);
    }
    return response;
  }


  /**
   * Admission control for Differentiated Service flows
   * @param   source  The node id of the source router
   * @param   destination  The node id of the destination router
   * @param   flow_request  The flow request message
   * @return  The flow response that contain the result of the
   * admission control.
      */
  public synchronized FlowResponse DS_Admission(FlowRequest
flow_request) {
    FlowResponse response;

    int source =
      PIB.findARouterOnLink(flow_request.getSourceInterface());

    // find a router on the same subnet as the destination host
```

```
  int destination =
    PIB.findARouterOnLink(flow_request.getDestinationInterface());

  response = DS_Admission(source, destination, flow_request);
  return response;
}

/**
 * Read data in SLSDbase and store inside various SLSTables
 */
private void setupSLSDbase() {
  BufferedReader bufReader;
  String slsData;
  StringTokenizer st;
  FileIO fileIO = new FileIO();
  fileIO.openToRead("server\\diffserv\\SLSDbase.dat");
  //read from database file
  String title = fileIO.readLine();
  //gui.sendText("SLSDbase.dat contains: ");
  //gui.sendText(title);
  slsData = fileIO.readLine();
  while (slsData != null) {
    st = new StringTokenizer(slsData);
    //gui.sendText(slsData);
    addSLSTable(st);
    //read next line of string from file
    slsData = fileIO.readLine();
  }
}


/**
 * Adds a SLSTable with the information given in the StringTokenizer
 * into the SLSDbase
 * @param   st
 */
private void addSLSTable(StringTokenizer st) {
  //create SLS Dbase with the information
  Integer nodeID = new Integer(st.nextToken());
  slsTable = slsDbase.getSLSTable(nodeID);
  while (st.hasMoreTokens()) {
    if (slsTable != null) {
    slsTable.addSLS(st.nextToken().hashCode(),      //user_id
                    st.nextToken());  //service class
    //slsDbase.displaySLSTable(nodeID);
    }
    else {
    slsTable = new SLSTable();
    //System.out.print("SLSTable created: ");
    slsTable.addSLS(st.nextToken().hashCode(),      //user_id
                    st.nextToken());  //service class
      slsDbase.addSLSTable(nodeID, slsTable);
    }
  }
}
```

```
  /**
   * Adds a SLS to the SLS_Dbase with the parameters given
   * @param   source  The node_id of the router
   * @param   flow_request  The flow request associated to the router
   * @return  The SLS object created from the parameters
   */
  private SLS addSLS(Integer source, FlowRequest flow_request) {
    SLS newSLS = new SLS(flow_request.getRequestedThroughput(),
                         flow_request.getRequestedLossRate(),
    flow_request.getRequestedDelay());
    SLSTable slsTable = slsDbase.getSLSTable(source);
    if (slsTable != null) {
      slsTable.addSLS(flow_request.getUser(), newSLS);
    }
    else {
      slsTable = new SLSTable();
      slsTable.addSLS(flow_request.getUser(), newSLS);
      slsDbase.addSLSTable(source, slsTable);
      gui.sendText("New SLSTable: " + slsTable.toString());
    }
    slsDbase.displaySLSTable();
    return newSLS;
  }


  /**
   * Remove the SLS from the SLSTable of the node_id and user_id
   * given in the parameters
   * @param   node_id  The node_id of the router
   * @param   user_id  The user_id of the user/application
   */
  private void deleteSLS(Integer node_id, int user_id) {
    SLSTable slsTable = slsDbase.getSLSTable(node_id);
    if (slsTable == null) {
      gui.sendText("SLSTable of node " + node_id + " not found");
    }
    else {
      slsTable.deleteSLS(user_id);
    }
    slsDbase.displaySLSTable();     //display SLSDbase
    controlExec.updateSLSTable();//only needed for displaying SLSTable
  }


//*********************************************************************
********
// These methods handle external network communications to routers
//*********************************************************************
*******/

  /**
   * Sends a flow routing table entry update message to a router. This
message
   * provides the router the required information to forward packets
based on
   * its flow id.
   * @param sourceAddress The router that will receive the FRTE update.
```

```
   * @param flow_id The id assigned to the flow in question.
   * @param next_hop The IPv6 address of the next node in the path.
   * @param service_level The service level that this flow is assigned
to.
   */
  public void sendFRTEUpdate(IPv6Address sourceAddress, int flow_id,
                                    IPv6Address next_hop, int
service_level) {
    FlowRoutingTableEntry myFRTE = new FlowRoutingTableEntry(flow_id,
                                                (byte)service_level,
next_hop);
    if(showComments){
      //gui.sendText("Server: sendFRTEUpdate: flowLookUp hashtable:");
      //gui.sendText(""+flowLookUp);
      gui.sendText("Server: sendFRTEUpdate: "+myFRTE);
    }
    if (serverType == 0) {///Primary Server
      int sourcePort = PSUEDORANDOMSOURCEPORT;
          //controlExec.listenToRandomPort(this);
      short destPort = ControlExecutive.SAAM_CONTROL_PORT;
      IPv6Address destHost = sourceAddress;
      // take steps to determine what flow id to send the packet on
      Vector interfaces = new Vector();
      interfaces.addElement(destHost);
      int destNodeId = PIB.doesRouterExist(interfaces);
      //int flowIdToSendItOn = ((Integer)flowLookUp.get
      //                        (new Integer(destNodeId))).intValue();
      int flowIdToSendItOn = getServerFlowId();
      try{
        controlExec.send(this,myFRTE, flowIdToSendItOn,
(short)sourcePort,
                         destHost, destPort);
      }catch (FlowException fe){
        System.err.println(fe.toString());
      }
      if (showComments){
        gui.sendText("Server: sendFRTEUpdate: FRTE for flow " + flow_id
          + " sent to interface "+sourceAddress);
        gui.sendText("          with next hop= "+next_hop
          +" on service level "+service_level+" via flow
"+flowIdToSendItOn);
      }
    }//end if serverType
    else {
      gui.sendText("I'm BackUp Server");
    }
  }


  /**
   * Sends a flow response to the requesting application to notify it
of
   * its newly assigned flow id. A flow id of zero is used to indicate
that the
   * flow cannot be supported. Once a flow response message is
instantiated and
```

```
   * a source and destination port is defined, the control executive's
send()
   * is called to send it to the destination host.
   * @param flow_request The flow request message that was received.
   * @param flow_id The flow id that is assigned to the flow request.
   */
  public void sendFlowResponse(FlowRequest flow_request, int flow_id){
    if(showComments){
      //gui.sendText("Server: sendFlowResponse: flowLookUp
hashtable:");
      //gui.sendText(""+flowLookUp);
      gui.sendText("Server: sendFlowResponse with flow_id:"+flow_id);
    }
    FlowResponse response = new
FlowResponse(flow_request.getTimeStamp(),
      flow_id);
    if (serverType == 0) {///Primary Server
      int sourcePort = PSUEDORANDOMSOURCEPORT;
              //controlExec.listenToRandomPort(this);
      short destPort = ControlExecutive.SAAM_CONTROL_PORT;
      IPv6Address destHost = flow_request.getSourceInterface();
      // take steps to determine what flow id to send the packet on
      Vector interfaces = new Vector();
      interfaces.addElement(destHost);
      //int destNodeId = PIB.doesRouterExist(interfaces);
      //int flowIdToSendItOn = ((Integer)flowLookUp.get
      //                         (new Integer(destNodeId))).intValue();
      int flowIdToSendItOn = getServerFlowId();
      try{
        controlExec.send(this, response, flowIdToSendItOn,
(short)sourcePort,
                         destHost, destPort);
      }catch(FlowException fe){
        System.err.println(fe.toString());
      }
      if (showComments){
        gui.sendText(""//"Server: sendFlowResponse: Flow response "
          + response + " from SourcePort: "+sourcePort+" to "+destHost
          + " sent via flow "+flowIdToSendItOn);
      }
    }//end if serverType
    else {
      gui.sendText("I'm BackUp Server");
    }
  }

  /**Henry
   * Sends a flow response to the requesting application to notify it
of
   * its newly assigned flow id. A flow id of zero is used to indicate
that the
   * flow cannot be supported. Once a flow response message is
instantiated and
   * a source and destination port is defined, the control executive's
send()
   * is called to send it to the destination host.
   * @param flow_request The flow request message that was received.
```

```
   * @param flow_id The flow id that is assigned to the flow request.
   */
  public void sendFlowResponse(FlowRequest flow_request,
                               int flow_id, byte result){
//modified by Henry for IntServ
    if(showComments){
      //gui.sendText("Server: sendFlowResponse: flowLookUp
hashtable:");
      //gui.sendText(""+flowLookUp);
      gui.sendText("Server: sendFlowResponse with flow_id:"+flow_id);
    }
    FlowResponse response = new
FlowResponse(flow_request.getTimeStamp(),
      flow_id);
    Vector data = new Vector();
    data.add(flow_request.getSourceInterface().toString());
    data.add(flow_request.getDestinationInterface().toString());
    data.add("IntServ");
    data.add(""+flow_request.getRequestedThroughput());
    data.add("Result: "+response.getResult());
    //data.add(""+flow_request.getUser());
    //data.add(""+response.getFlowId());
    flowTableData.add(data);
    controlExec.updateFlowTable(flowTableData);
    if (serverType == 0) {///Primary Server
      int sourcePort = PSUEDORANDOMSOURCEPORT;
                       //controlExec.listenToRandomPort(this);
      short destPort = ControlExecutive.SAAM_CONTROL_PORT;
      IPv6Address destHost = flow_request.getSourceInterface();
      // take steps to determine what flow id to send the packet on
      //Vector interfaces = new Vector();
      //interfaces.addElement(destHost);
      //int destNodeId = PIB.doesRouterExist(interfaces);
      //int flowIdToSendItOn = ((Integer)flowLookUp.get
      //                        (new Integer(destNodeId))).intValue();
      int flowIdToSendItOn = getServerFlowId();
      try{
        controlExec.send(this, response, flowIdToSendItOn,
                  (short)sourcePort, destHost, destPort);
      }catch(FlowException fe){
        System.err.println(fe.toString());
      }
      if (showComments){
        gui.sendText("Server: sendFlowResponse: Flow response "+
response);
        gui.sendText("  with length = "+response.length()
           +" from SourcePort: "+sourcePort+" to "+destHost
           + " sent via flow "+flowIdToSendItOn);
      }
    }//end if serverType
    else {
      gui.sendText("I'm BackUp Server");
    }
  }//end sendFlowResponse

//Added by Henry
  /**
```

```
   * Sends a flow response to the requesting application to notify it
of
   * its newly assigned service level spec (SLS). Once a flow response
   * message is instantiated and a source and destination port is
defined,
   * the control executive's send() is called to send it to the
destination host.
   * @param flow_request The flow request message that was received.
   * @param result The outcome of the admission control to the flow
request.
   */
  public void sendFlowResponse(FlowRequest flow_request, byte result,
SLS newSLS){
            FlowResponse response;
    //create a response message for DiffServ
    if (newSLS == null) {
      response = new FlowResponse(flow_request.getTimeStamp(), result);
    }
    else {
      response = new FlowResponse(flow_request.getTimeStamp(),
                    result, flow_request.getUser(), newSLS);
    }
    if(showComments){
      //gui.sendText("Server: sendFlowResponse: flowLookUp
hashtable:");
      //gui.sendText(""+flowLookUp);
      gui.sendText("Server: sendFlowResponse: "+response);
    }
    Vector data = new Vector();
    data.add(flow_request.getSourceInterface().toString());
    data.add(flow_request.getDestinationInterface().toString());
    data.add("DiffServ");
    data.add(""+flow_request.getRequestedThroughput());
    data.add("Result: "+response.getResult());
    //data.add(""+flow_request.getUser());
    //data.add(""+response.getFlowId());
    flowTableData.add(data);
    controlExec.updateFlowTable(flowTableData);
    if (serverType == 0) {///Primary Server
      int sourcePort = PSUEDORANDOMSOURCEPORT;
           //controlExec.listenToRandomPort(this);
      short destPort = ControlExecutive.SAAM_CONTROL_PORT;
      IPv6Address destHost = flow_request.getSourceInterface();
      // take steps to determine what flow id to send the packet on
      //Vector interfaces = new Vector();
      //interfaces.addElement(destHost);
      //int destNodeId = PIB.doesRouterExist(interfaces);
      //int flowIdToSendItOn = ((Integer)flowLookUp.get
        //                      (new Integer(destNodeId))).intValue();
      int flowIdToSendItOn = getServerFlowId();
      try{
        controlExec.send(this, response, flowIdToSendItOn,
                    (short)sourcePort, destHost, destPort);
      }
      catch(FlowException fe){
        System.err.println(fe.toString());
      }
```

```
      if (showComments){
        gui.sendText("Server: sendFlowResponse: Flow response "
          + response + " from SourcePort: "+sourcePort+" to "+destHost
          + " sent via flow "+flowIdToSendItOn);
      }
    }//end if serverType
    else {
      gui.sendText("I'm BackUp Server");
    }
  }//end sendFlowResponse

  /**by Henry
   * Sends a resource allocation message to the router with the address
   * specified in the parameter to initialize the amount of resources
it
   * has been allocated for its service level pipes.
   */
  public void initializeResourceAllocation(IPv6Address node_id){
    int routerID;
    if (showComments){
      gui.sendText("Server: initializeResourceAllocation ");
    }
    int[] allocated_throughput = new int[NUMBEROFSERVICELEVELS];
    for (int i=0; i<allocated_throughput.length; i++) {
      allocated_throughput[i] =
(int)(throughputRatioForSL[i]*INITIALTHROUGHPUT);
    }
    sendResourceAllocation(node_id, allocated_throughput);
    //Vector routerIDs = PIB.getAllRouterIds();
    for (int i=0; i<allocated_throughput.length; i++) {
      //routerID = ((Integer)routerIDs.get(i)).intValue();
      routerID =
((Integer)IPv6ToIntIdTable.get(node_id.toString())).intValue();
      Vector interfaceIDs = PIB.getRouterInterfaces(routerID);
      gui.sendText("routerID = "+routerID+" has interface:
"+interfaceIDs);
      for (int j=0; j<interfaceIDs.size(); j++) {
        IPv6Address address = (IPv6Address)interfaceIDs.get(j);
        PIB.updateSLP(address, i, INITIALDELAY, INITIALLOSSRATE,
            //INITIALTHROUGHPUT);
            allocated_throughput[i]);
      }
    }
  }//end initializeResourceAllocation


  /**by Henry
   * Sends a resource allocation message to the router to update the
   * amount of resources it has been allocated for its service level
pipes.
   * @param  destination  The IPv6Address of the router
   * @param  allocated_throughput  The amount of resources it has been
   * allocated for its service level pipes.
   */
  public void sendResourceAllocation(IPv6Address destination,
                                     int[] allocated_throughput){
```

```
    if (serverType == 0) {///Primary Server
    ResourceAllocation myRA = new
ResourceAllocation(allocated_throughput);
    int sourcePort = PSUEDORANDOMSOURCEPORT;
                    //controlExec.listenToRandomPort(this);
    short destPort = ControlExecutive.SAAM_CONTROL_PORT;
    IPv6Address destHost = destination;
    int flowIdToSendItOn = getServerFlowId();
    if (showComments){
      gui.sendText("Server: sendResourceAllocation: RA " +myRA+
            " sent to interface "+destination+" via flow
"+flowIdToSendItOn);
    }
    try{
      controlExec.send(this, myRA, flowIdToSendItOn,
                (short)sourcePort, destHost, destPort);
    }
    catch (FlowException fe){
      System.err.println(fe.toString());
    }
   }//end if
  }//end sendResourceAllocation


  /**by Henry
   * Sends the SLS information contained in the SLSTable of the router
   * specified in the parameter to it using SLSTableEntry messages
   * @param   routerId  The IPv6Address of the router
   * @param   nodeID  The node_id of the router
   */
  private void sendSLSTable(IPv6Address routerId, Integer nodeID) {
    IPv6Address destHost = routerId;
    slsTable = slsDbase.getSLSTable(nodeID);
    if (serverType == 0) {///Primary Server
      int sourcePort = PSUEDORANDOMSOURCEPORT;
                      //controlExec.listenToRandomPort(this);
      short destPort = ControlExecutive.SAAM_CONTROL_PORT;
      int flowIdToSendItOn = getServerFlowId();
      if (showComments){
        gui.sendText("Server: sendSLSTable to "+destHost+
            " via flow "+flowIdToSendItOn);
      }
      Enumeration e = slsTable.keys();
      // for each of the user in the SLSTable
      while(e.hasMoreElements()){
        Integer user_id = (Integer)e.nextElement();
        SLS sls = slsTable.getSLS(user_id.intValue());
        SLSTableEntry mySLSMessage = new
SLSTableEntry(user_id.intValue(), sls);
        try{
          controlExec.send(this, mySLSMessage, flowIdToSendItOn,
                  (short)sourcePort, destHost, destPort);
           Thread.sleep(1000);
        }
        catch (FlowException fe){
          System.err.println(fe.toString());
        }
```

```
        catch(InterruptedException ie){
        }
      }//end while
    }//end if
  }//end sendSLSTable


  /**Henry
   * Sends a SLS to the router specified in the parameter using a
   * SLSTableEntry message to update its SLSTable
   * @param   routerId  The IPv6Address of the router
   * @param   user_id  The user/application that has been assigned the
SLS
   * @param   sls  The SLS that is being assigned to the
user/application
   */
  private void sendSLSTableEntry(IPv6Address routerId, int user_id, SLS
sls){
    IPv6Address destHost = routerId;
    if (serverType == 0) {///Primary Server
      int sourcePort = PSUEDORANDOMSOURCEPORT;
                       //controlExec.listenToRandomPort(this);
      short destPort = ControlExecutive.SAAM_CONTROL_PORT;
      int flowIdToSendItOn = getServerFlowId();
      if (showComments){
        gui.sendText("Server: sendSLSTableEntry to "+destHost+
            " via flow "+flowIdToSendItOn);
      }
      SLSTableEntry mySLSMessage = new SLSTableEntry(user_id, sls);
      try{
        controlExec.send(this, mySLSMessage, flowIdToSendItOn,
                (short)sourcePort, destHost, destPort);
         Thread.sleep(1000);
      }
      catch (FlowException fe){
        System.err.println(fe.toString());
      }
      catch(InterruptedException ie){
      }
    }//end if
  }

  /**Henry   (not used at the moment)
   * Sends a SLSTableEntry message to the all the routers to update the
   * SLSTable it has for its differentiated service level pipes.
   * @param flow_request The flow request message that was received.
   * @param result The outcome of the admission control to the flow
request.
   */
  public void sendSLSMessage(IPv6Address[] pathAddress, int user_id,
SLS sls){
    SLSTableEntry mySLSMessage = new SLSTableEntry(user_id, sls);
    if (serverType == 0) {///Primary Server
      int sourcePort = PSUEDORANDOMSOURCEPORT;
                       //controlExec.listenToRandomPort(this);
      short destPort = ControlExecutive.SAAM_CONTROL_PORT;
      for (int i=0; i<pathAddress.length; i++) {
```

```
        IPv6Address destHost = pathAddress[i];
        // take steps to determine what flow id to send the packet on
        Vector interfaces = new Vector();
        interfaces.addElement(destHost);
        int destNodeId = PIB.doesRouterExist(interfaces);
        //int flowIdToSendItOn = ((Integer)flowLookUp.get
          //                      (new Integer(destNodeId))).intValue();
            int flowIdToSendItOn = getServerFlowId();
        try{
          controlExec.send(this,mySLSMessage, flowIdToSendItOn,
                  (short)sourcePort, destHost, destPort);
        }
        catch (FlowException fe){
          System.err.println(fe.toString());
        }
      }
      /*
      if (showComments){
        gui.sendText("Server: sendSLSMessage " +
          " to interface "+destination);
        gui.sendText(" via flow "+flowIdToSendItOn);
      } */
    }//end if
  }//end sendSLSMessage


//**********************************************************************
********
// These methods handle internal manipulation of data describing
network status
//**********************************************************************
*******/

  /**
   * Determines all of the possible paths that exist between any source
and
   * destination router in the network. This determination is based on
the
   * physical definition of the network that is provided by the hello
messages
   * received from the routers and stored within the PIB. The paths
that are
   * found are then recorded in the PIB for fast assignment of flows
later.<p>
   * All node ids are first retrieved from the PIB. For each service
level, we
   * build an array of parents of each node. A parent is node that is
directly
   * connected. Those directly connected nodes would have service level
pipes
   * that would need to be passed through to get to the child node in
question.
   * This parent array is used to populate a path table. Each node id
is
   * assigned as the final destination of path and all of the different
paths
```

```
   * are then found by working out from this destination. For each of
these
   * destination nodes, a call is made to processPath() to find all the
valid
   * paths that go to this destination node. We make the call with a
specified
   * height of search of 1.
   */
  public void findAllPossiblePaths() {
    long start, finish;
    int NumberOfRouters;
    int max_slp_id = INITIALZERO;
    /** A count of the highest path id assigned so far. */
    int max_path_id = INITIALZERO;
    int service_level = INITIALZERO;

    /** A vector of the routers that are known by the db. */
    Vector V = new Vector();

    /** A vector of the parent routers for each given destination
router. */
    Hashtable parent;

    // capture the start time of processing a path data
    start = System.currentTimeMillis();

    // reset the maximum path id assigned so far to zero
    max_path_id = INITIALPATHID;

    V = PIB.getAllRouterIds();
    //gui.sendText("Server: findAllPossiblePaths: has routers =
"+V.toString());

    //retrieve COUNT of routers
    NumberOfRouters = V.size();

    //find all possible paths for each service level
    //max_slp_id = (new Integer(PIB.findMaxServiceLevel())).intValue();
    max_slp_id = NUMBEROFSERVICELEVELS;
    //gui.sendText("Server: findAllPossiblePaths: has max_slp_id =
"+max_slp_id);

    for (service_level = INITIALZERO; service_level < max_slp_id;
service_level++){

      //build parent array of each SLP at this service level
      parent = PIB.getParents(V, service_level);
    //gui.sendText("Server: findAllPossiblePaths: has parents =
"+parent.toString());

      //populate path table
      for (int index = INITIALZERO; index < NumberOfRouters; index++){
        int heightOfSearch = INITIALHEIGHTOFSEARCH;
        int aPath[] = new int[Hmax + INCREMENTATIONOFSEARCH];
        aPath[DESTINATIONNODE] =
((Integer)V.elementAt(index)).intValue();
        processPath(parent, aPath, heightOfSearch,service_level);
```

```
      }
    }

    // capture the path data processing finish time
    finish = System.currentTimeMillis();
    gui.sendText("Server: findAllPossiblePaths: Time required = "
      +(finish-start)+" milliseconds.");

    timeOfLastPIBBuild = finish;

  }

  /**
   * Processes all valid paths that arrive at the destination node
within some
   * range of hops. For each parent of the node at the distance of
   * heightOfSearch from the destination, a check is made to ensure
that adding
   * this new parent will cause no cycle. If this checks out, then that
parent
   * can be added and a new path can be assigned. The service level
pipes in
   * this new path are identified and their sequence numbers in this
path are
   * recorded to the PIB. Next, a check is made to see if the height of
the
   * search is less than the server's max search height of Hmax. If it
is less,
   * the method recursively calls itself with an incremented
heightOfSearch
   * variable.
   * @param parent Contains each router and a list of other
   * routers that are directly attached to them.
   * @param aPath[] An array contain a path from a source node,
   * aPath[heightOfSearch], to a destination node, aPath[0].
   * @param heightOfSearch The number of nodes in the path so far.
   * @param service_level The level of service assigned to a flow.
   */

  public void processPath(Hashtable parent,
                          int aPath[], int heightOfSearch, int
service_level){
    IPv6Address link_id;
    int justARouter;
    int sequence_number;
    int path_id;
    Enumeration W = ((Vector)parent.get(
                            new Integer(aPath[heightOfSearch-
1]))).elements();
    while (W.hasMoreElements()) {
      justARouter = ((Integer)W.nextElement()).intValue();
      if (causeNoCycle(aPath, heightOfSearch, justARouter)) {

        // assign this router as the source in this path
        aPath[heightOfSearch] = justARouter;

        // record the new path id, etc.
```

```
        path_id = PIB.getNewPathId(justARouter,
aPath[DESTINATIONNODE]);

        // run through the SLP's and record their sequence
        for (int index = heightOfSearch; index >DESTINATIONNODE; index-
-){

          // determine link_id of this SLP
          link_id = PIB.getLinkBetween(aPath[index],
                                       aPath[index-
INCREMENTATIONOFSEARCH]);

          // assign the SLP its sequence number
          sequence_number = heightOfSearch - index;
          PIB.assignSLPSequence(service_level, aPath[index],
            link_id, path_id, sequence_number);
        }
        if (heightOfSearch < Hmax) {
          processPath(parent, aPath,
heightOfSearch+INCREMENTATIONOFSEARCH,

service_level);
        }
      }
    }
    if (showComments){
      gui.sendText("Server: processPath: paths at depth of
"+heightOfSearch
        +" from node "+aPath[DESTINATIONNODE]+" is completed.");
    }
  }

  /**
   * Checks to ensure that the addition of a specified new node to a
specified
   * path does not result in a cycle being created. This check is
completed by
   * the new node is already a member of the list of nodes in the path
already.
   * @param aPath[] An array contain a path from a source node,
   * aPath[heightOfSearch], to a destination node, aPath[0].
   * @param heightOfSearch The number of nodes in the path so far.
   * @param justARouter The proposed next node in for a new path.
   * @returns noCycles True if no cycles are created by the addition of
   * justARouter.
   */
  public boolean causeNoCycle(int aPath[], int heightOfSearch,
                              int justARouter){
    boolean noCycles = true;
    for (int index = INITIALZERO; index < heightOfSearch; index++){
      if (justARouter == aPath[index]){
        if (showComments){
          gui.sendText("Server: causeNoCycle: adding "+justARouter
            +" to get to "+aPath[DESTINATIONNODE]+" via "
            +aPath[heightOfSearch-INCREMENTATIONOFSEARCH]
            +" at a height of "+heightOfSearch+" caused cycle!");
        }
```

```
        return noCycles = false;
      }
    }
    if (showComments){
      gui.sendText("Server: causeNoCycle: adding "+justARouter
        +" as hop #"+heightOfSearch+" to get to
"+aPath[DESTINATIONNODE]
        +" via "+aPath[heightOfSearch-INCREMENTATIONOFSEARCH]
        +" does not cause cycle.");
    }
    return noCycles;
  }

  /**
   * Determines what the effective QoS on each path in the PIB is. For
each
   * path, the service level pipes that compose it are retrieved. Then,
for
   * each of these service level pipes, we total up the delay and loss
rate.
   * The effective throughput remaining is determined by finding the
minimum
   * difference between the observed throughput and the target
throughput of
   * each service level pipe.
   */
  public void determineEffectiveQoSForPaths(){
    long start, finish;
    Vector path_ids;
    Integer myPathId;
    Vector SLPs;
    SLP mySLP;
    int totalDelay = INITIALZERO, totalLossRate = INITIALZERO,
      throughput = INITIALZERO, targetThroughput = INITIALZERO,
      throughputRemaining = INITIALZERO,
      minThroughputRemaining = INITIALZERO;

    // capture the start time of processing a path data
    start = System.currentTimeMillis();

    // for each path
    path_ids = PIB.getAllPathIds();
    //gui.sendText("determineEffectiveQoSForPaths: allPathIds =
"+path_ids);

    for (int index1 = INITIALZERO; index1 < path_ids.size(); index1++){

      // for each path
      myPathId = (Integer)path_ids.elementAt(index1);
      //gui.sendText("myPathId: "+myPathId);

      //SLPs = PIB.getSLPsOfPath(myPathId.intValue());
      SLPs = PIB.getSLPsOfAPath(myPathId.intValue());
      //gui.sendText("SLPs: "+SLPs);

      for (int index2 = INITIALZERO; index2 < SLPs.size(); index2++){
        //gui.sendText("Taking the slp "+index2);
```

```
        mySLP = (SLP)SLPs.elementAt(index2);
        //gui.sendText("slp is taken");
        // add delay to total delay
        totalDelay = totalDelay + mySLP.getDelay();
        //gui.sendText("delay is taken"+totalDelay);
        // add loss rate to total loss rate
        totalLossRate = totalLossRate + mySLP.getLossRate();

        // find min throughput
        throughput = mySLP.getThroughput();

        targetThroughput = mySLP.getAllocatedThroughput();

        throughputRemaining = targetThroughput - throughput;

        if (throughputRemaining < minThroughputRemaining ||
                                  minThroughputRemaining == INITIALZERO){
          minThroughputRemaining = throughputRemaining;
        }

      }

      //gui.sendText("setEffectiveQoSOfPath with minThroughputRemaining
= "
      //            +minThroughputRemaining);

      PIB.setEffectiveQoSOfPath(myPathId.intValue(),
            totalDelay, totalLossRate, minThroughputRemaining);

      totalDelay = INITIALZERO;
      totalLossRate = INITIALZERO;
      minThroughputRemaining = INITIALZERO;
    }

    // capture the path data processing finish time
    finish = System.currentTimeMillis();
    gui.sendText("Server: determineEffectiveQoSForPaths: Time required
= "
      +(finish-start)+" milliseconds.");

  }

  /**
   * Determines the effective QoS for just those paths that pass over
the
   * specified service level pipe. For each path, the service level
pipes that
   * compose it are retrieved. Then, for each of these service level
pipes, we
   * total up the delay and loss rate. The effective throughput
remaining is
   * determined by finding the minimum difference between the observed
   * throughput and the target throughput of each service level pipe.
   * @param address The address of the interface containing this
service level.
   * @param service_level The service level of this SLP.
   */
```

```
  public void determineEffectiveQoSForPaths(IPv6Address address, int
service_level){
    long start, finish;
    Vector path_ids;
    Integer myPathId;
    Vector SLPs;
    SLP mySLP;
    int totalDelay = INITIALZERO, totalLossRate = INITIALZERO,
      throughput = INITIALZERO, targetThroughput = INITIALZERO,
      throughputRemaining = INITIALZERO, minThroughputRemaining =
INITIALZERO;

    // capture the start time of processing a path data
    start = System.currentTimeMillis();

    // for each path
    path_ids = PIB.getAllPathIdsThatTraverseSLP(address,
service_level);

    for (int index1 = INITIALZERO; index1 < path_ids.size(); index1++){

      // for each link
      myPathId = (Integer)path_ids.elementAt(index1);

      //SLPs = PIB.getSLPsOfPath(myPathId.intValue());
      SLPs = PIB.getSLPsOfAPath(myPathId.intValue());

      for (int index2 = INITIALZERO; index2 < SLPs.size(); index2++){

        mySLP = (SLP)SLPs.elementAt(index2);

        // add delay to total delay
        totalDelay = totalDelay + mySLP.getDelay();

        // add loss rate to total loss rate
        totalLossRate = totalLossRate + mySLP.getLossRate();

        // find min throughput
        throughput = mySLP.getThroughput();

        targetThroughput = mySLP.getAllocatedThroughput();

        throughputRemaining = targetThroughput - throughput;
        if (throughputRemaining < minThroughputRemaining ||
                                        minThroughputRemaining ==
INITIALZERO){
          minThroughputRemaining = throughputRemaining;
        }

      }


PIB.setEffectiveQoSOfPath(myPathId.intValue(),totalDelay,totalLossRate,

minThroughputRemaining);
      totalDelay = INITIALZERO;
```

```
      totalLossRate = INITIALZERO;
      minThroughputRemaining = INITIALZERO;
    }

    // capture the path data processing finish time
    finish = System.currentTimeMillis();
    gui.sendText("Server: determineEffectiveQoSForPaths: Time required
="
      +(finish-start)+" milliseconds.");
  }

  /**
   * Returns the String representation of this Server.
   * @return The String representation of this Server.
   */
  public String toString(){
    return "Server";
  }

  //methods below are added by akkoc
  /**
   * Creates thread for dcm sending from the server.
   * @return void.
   */
  public void autoConfig() {
    configThread = new Thread(this,"AutoConfig");
    configThread.start();
  }//end of autoconfig

  /**
   * Triggers DCM sending. and provides continues resreshment of SAAM
region
   * with DCM messages.
   * @return void.
   */
  public void run(){
    gui.sendText("\n Server  will send first DCM after 60 secs");
    System.out.println("\n Server  will send first DCM after 60 secs");
    try{
      //gui.sendText("thread is sleeping now ");
      configThread.sleep(30000);
      //gui.sendText("thread woke up after 30  secs so start sending
");
      System.out.println("thread woke up after 50  secs so start
sending ");
    }catch(InterruptedException ie){}

 //while(true) {

   try{
     Vector tableEntries =
controlExec.getEmulationTable().getEmTable();
      System.out.println(" Emulatin table ok ");
     Enumeration es = tableEntries.elements();
     while( es.hasMoreElements()){
       EmulationTableEntry ent = (EmulationTableEntry)
es.nextElement();
```

```
        //destination adress determined from emulationtable entry
        IPv6Address des = new
IPv6Address(ent.getNextHopIPv6().getAddress());
        gui.sendText(" Destination of DCM is "+des.toString());
        System.out.println(" Destination of DCM is "+des.toString());
        byte[] nextHopBytes = des.getAddress();
        Vector interfaces = new Vector();
        interfaces = this.controlExec.getInterfaces();

        IPv6Address sInt;
        for(int i=0;i<interfaces.size();i++){
          Interface thisInterface = (Interface)interfaces.get(i);
          //cycle through all interfaces checking network address
against nextHop.
          int match = 0;
          byte[] outboundInterfaceBytes =
thisInterface.getID().getIPv6().getAddress();
          int bytesToCheck = 5;

          for(int index=0;index<bytesToCheck;index++){
            if((nextHopBytes[index]&0xFF)==
(outboundInterfaceBytes[index]&0xFF)){
               match++;
            }//if
          }//inner for

          if(match== bytesToCheck){
            sInt = new
IPv6Address(thisInterface.getID().getIPv6().getAddress());
            sendDown(sInt,des);
          }//if
        }//outer for

      }// end while
    }catch(UnknownHostException e){
      gui.sendText(e.getMessage()+"inside catch of DCM start up using em
table  ");
    }//try-catch

    try{
      Thread.sleep(this.cycleTime); //from demostation
    }catch(InterruptedException ie){
      gui.sendText("thread sleep problem");
    }

       //}//end of while providing continues DCM sending

  }// end run()


   /**
    * Retruns flowid of server.
    * @return ind serverflow id.
    */
   public int getServerFlowId(){
     return flowId;
   }
```

```
  /**
   * Returns type of server(0-> for Primary, 1-> for Backup )
   * @return byte value.
   */
  public byte  getServerType(){
     return serverType;
  }

  /**
   * Method to send the DCM message using controlExecutive sendDCM
method
   * @return void.
   */
  public void sendDown(IPv6Address srcInt,IPv6Address des) {

    DCM myDCM = new
DCM(flowId,ServerId,metricType,srcInt,CTS,globalTime,

getSeguenceNumberForDcmSending());
    gui.sendText("DCM with SQ is sent
"+this.getSeguenceNumberForDcmSending());
    setSeguenceNumberForDcmSending();
    short sourcePort = ControlExecutive.SAAM_CONTROL_PORT;
    short destPort = ControlExecutive.SAAM_CONTROL_PORT;

    try{
       controlExec.sendDCM(this, myDCM, getServerFlowId(),
sourcePort,des, destPort);
       gui.sendText("DCM has been sent");
    }catch(Exception fe){
       System.err.println(fe.toString());
    }

  }//end sendDown()

  /**
   * Method for setting proper value to put in DCM message for sequence
   * number field
   * @return void.
   */
  private void setSeguenceNumberForDcmSending(){
    sequenceNumber++;
    if(sequenceNumber == 65535) sequenceNumber = 0;
  }

   /**
   * Method for returning current sequence number value
   * @return int value.
   */

  private int getSeguenceNumberForDcmSending(){
    return  sequenceNumber;
  }


 /**
```

```
   * Method for receiving required values from demosation for server
settings
   * Also this method is used for server to place an entry for itself
   * in the servertable
   * @return void.
   */
  public synchronized void processConfiguration (Configuration con){
    System.out.println("Inside server processCONFIGURATION ");
    serverType = con.getServerType();
    flowId = con.getFlowId();
    metricType = con.getmetricType();
    cycleTime = con.getCycleTime();
    globalTime = con.getGlobalTime();

    AutoConfigurationExecutive ace =
controlExec.getAutoConfigurationExecutive();
    ace.createNewServerInformation(flowId,controlExec.getRouterId());
    System.out.println("Process of the Configuration message is OK.");
    autoConfig();
  }// end processConfigurtaion

}//end of Server class
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C – SAAM MESSAGE.RESOURCEALLOCATION CLASS CODE

```
//9Feb2000[Henry]        - Modified
//13Dec99[Henry]         - Created

package saam.message;

import java.net.UnknownHostException;
import saam.net.*;
import saam.util.*;


/**
 * A ResourceAllocation message to allocate resouces for various
 * service level pipes.
 */
public class ResourceAllocation extends Message{

  /** The number of service level to be allocated for  */
  private byte numberOfSL = 0;

  /** The byte array which stores the message parameters  */
  private byte[] bytes;

  /** The integer array which stores the message parameters  */
  private int[] service_allotment;

  /**
   * No-args constructor used by the server.
   */
  public ResourceAllocation(){
    super(Message.RESOURCEALLOCATION_TYPE);
  }

  /**
   * Constructs a ResourceAllocation message with the parameters
   * supplied.
   * @param allotment The array of allocated throughput associated
   * with this Message.
   */
  public ResourceAllocation(int[] allotment) {

    super(Message.RESOURCEALLOCATION_TYPE);
    this.service_allotment = allotment;
    this.numberOfSL = (byte)allotment.length;

    for (int i=0; i<numberOfSL; i++){
      bytes = Array.concat(bytes,
          PrimitiveConversions.getBytes(allotment[i]));
    }
  }

  /**
   * Construct this Message from a byte array that is presumed
   * to conform to the proper format for this Message.  Presumably,
```

```
 * this constructor is called when the receiving PacketFactory
 * gets the byte array that represents this Message - a byte
 * array that was presumably generated when the sender of this
 * Message called the getBytes() method after creating this
 * Message and before sending it.
 */
public ResourceAllocation(byte[] bytes) {

  super(Message.RESOURCEALLOCATION_TYPE);
  this.bytes = bytes;
  int pointer=0;
  int index = 0;
  this.numberOfSL = (byte)(bytes.length/4);
  this.service_allotment = new int[numberOfSL];
  while (pointer<bytes.length) {
        service_allotment[index++] = PrimitiveConversions.getInt(
          Array.getSubArray(bytes,pointer, pointer+4));
        pointer+=4;
  }
}

/**
 * Returns the service allotment associated with this event.
 * @return The service allotment associated with this event.
 */
public int[] getServiceAllotment(){
  return service_allotment;
}

/**
 * Returns The byte array representation of this Message.
 * @return The byte array representation of this Message.
 */
public byte[] getBytes(){
  return bytes;
}

/**
 * Returns the number of service levels of this Message.
 * @return The number of service levels of this Message.
 */
public byte getNumOfServiceLevels(){
  return numberOfSL;
}

/**
 * Returns the length of this Message.
 * @return The length of this Message.
 */
public short length(){
  try{
    return (short)bytes.length;
  }
  catch(NullPointerException npe){
    return 0;
  }
}
```

```
  /**
   * Returns a <code>String</code> representation of this Message.
   * @return The <code>String</code> representation of this Message
   */
  public String toString(){
    String service_allocated =
          "ResourceAllocation for the various slps are:\n";

    for (int i=0; i<service_allotment.length; i++){
      service_allocated = service_allocated+"Service Level "+i
        +" = "+service_allotment[i]+"\n";
    }
    return service_allocated;
  }

}//end of ResourceAllocation class
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D – SAAM MESSAGE.FLOWREQUEST CLASS CODE

```
//14Dec99[Henry]   - Added declaration for service_level,
//                   and new constructors that assigns
//                   value to it.
//01Aug99[Dean]    - Created..

package saam.message;

import java.net.UnknownHostException;
import saam.net.*;
import saam.util.*;
import saam.server.diffserv.*;
import saam.server.*;


/**
 * An Object desiring to communicate within a SAAM network
 * will call the requestFlow method in the ControlExecutive
 * The ControlExecutive will then construct a FlowRequest
 * Message and send it to the server.
 */
public class FlowRequest extends Message{

  /** The address of the sender  */
  private IPv6Address source_interface = new IPv6Address();
  /** The address of the receiver  */
  private IPv6Address destination_interface = new IPv6Address();

  /** The level of service negotiated  */
  private byte service_level = Server.IS_SERVICELEVEL;

  /** The average delay negotiated  */
  private int requested_delay = 0;
  /** The average rate of packet loss negotiated. */
  private int requested_loss_rate = 0;
  /** The rate of data negotiated. */
  private int requested_throughput = 0;

  /** The service level spec for the flow. */
  private SLS sls;

  /** The hashcode that represent the user of this SLS */
  private int user_id = 0;

  /** The byte array which stores the message parameters  */
  private byte[] bytes;

  /** The time when this message is created  */
  private long time_stamp;

  /** The byte length of an IntServ FlowRequest  */
  private static final int INTSERV_SIZE = 53;

  /**
```

```
 * No-Args constructor which constructs a FlowRequest using
 * the default values for all fields. <p>
 * source_interface      = IPv6Address.DEFAULT_HOST;
 * destination_interface = IPv6Address.DEFAULT_HOST;
 * requested_delay       = 0;
 * requested_loss_rate   = 0;
 * requested_throughput  = 0;
 * etc.....
 */
public FlowRequest(){
  super(Message.FLOWREQUEST_TYPE);
  time_stamp = System.currentTimeMillis();
  sls = new SLS(requested_throughput,
      requested_loss_rate, requested_delay);
}

/**
 * Constructs a IntServ FlowRequest using the parameters supplied.
 * @param source_interface The IPv6Address of the source.
 * @param destination_interface The IPv6Address of the destination.
 * @param time_stamp The 8 byte time stamp.
 * @param requested_delay The maximum delay reuested.
 * @param requested_loss_rate The maximum loss rate requested.
 * @param requested_throughput The maximum throughput requested.
 */
public FlowRequest(IPv6Address source_interface,
  IPv6Address destination_interface,
  long time_stamp,
  int requested_delay,
  int requested_loss_rate,
  int requested_throughput){

  //set all instance variables
  this(source_interface, destination_interface,
      Server.IS_SERVICELEVEL, time_stamp, requested_delay,
        requested_loss_rate, requested_throughput);
}

/**
 * Constructs a FlowRequest using the parameters supplied.
 * @param source_interface The IPv6Address of the source.
 * @param destination_interface The IPv6Address of the
 * destination.
 * @param time_stamp The 8 byte time stamp.
 * @param requested_delay The maximum delay reuested.
 * @param requested_loss_rate The maximum loss rate
 * requested.
 * @param requested_throughput The maximum throughput
 * requested.
 */
public FlowRequest(IPv6Address source_interface,
  IPv6Address destination_interface,
  byte service_level,
  long time_stamp,
  int requested_delay,
  int requested_loss_rate,
  int requested_throughput){
```

```
    //set all instance variables
    super(Message.FLOWREQUEST_TYPE);
    this.source_interface = source_interface;
    this.destination_interface = destination_interface;
    this.service_level = service_level;
    this.time_stamp = time_stamp;
    this.requested_delay = requested_delay;
    this.requested_loss_rate = requested_loss_rate;
    this.requested_throughput = requested_throughput;
      convertToBytes(source_interface, destination_interface,
       service_level, time_stamp, requested_delay,
         requested_loss_rate, requested_throughput);
  }

  /**
   * Constructs a DiffServ FlowRequest using the parameters
   * supplied.
   * @param source_interface The IPv6Address of the source.
   * @param destination_interface The IPv6Address of the
   * destination.
   * @param time_stamp The 8 byte time stamp.
   * @param user_id The user identification number.
   * @param requested_delay The maximum delay reuested.
   * @param requested_loss_rate The maximum loss rate requested.
   * @param requested_throughput The maximum throughput requested.
   */
  public FlowRequest(IPv6Address source_interface,
    IPv6Address destination_interface,
    long time_stamp,
    int user_id,
    int requested_delay,
    int requested_loss_rate,
    int requested_throughput){

    //set all instance variables
    super(Message.FLOWREQUEST_TYPE);
    this.source_interface = source_interface;
    this.destination_interface = destination_interface;
    this.service_level = Server.DS_SERVICELEVEL;
    this.time_stamp = time_stamp;
    this.user_id = user_id;
    this.sls = new SLS(requested_delay,
         requested_loss_rate, requested_throughput);
    this.requested_delay = sls.getDelay();
    this.requested_loss_rate = sls.getLossRate();
    this.requested_throughput = sls.getProfile();
    convertToBytes(source_interface, destination_interface,
        service_level,time_stamp, user_id, sls);
  }

  /**
   * Constructs a DiffServ FlowRequest using the parameters
   * supplied.
   * @param source_interface The IPv6Address of the source.
   * @param destination_interface The IPv6Address of the
   * destination.
```

```
   * @param time_stamp The 8 byte time stamp.
   * @param user_id The user identification number.
   * @param sls The type of SLS requested for.
   */
  public FlowRequest(IPv6Address source_interface,
    IPv6Address destination_interface,
    long time_stamp,
    int user_id,
    SLS sls) {

    //set all instance variables
    super(Message.FLOWREQUEST_TYPE);
    this.source_interface = source_interface;
    this.destination_interface = destination_interface;
    this.service_level = Server.DS_SERVICELEVEL;
    this.time_stamp = time_stamp;
    this.user_id = user_id;
    this.sls = sls;
    this.requested_delay = sls.getDelay();
    this.requested_loss_rate = sls.getLossRate();
    this.requested_throughput = sls.getProfile();
    convertToBytes(source_interface, destination_interface,
        service_level, time_stamp, user_id, sls);
  }

  /**
   * Construct this Message from a byte array that is presumed to
conform
   * to the proper format for this Message.  Presumably, this
constructor
   * is called when the receiving PacketFactory gets the byte array
that
   * represents this Message - a byte array that was presumably
generated
   * when the sender of this Message called the getBytes() method after
   * creating this Message and before sending it.
   */
  public FlowRequest(byte[] bytes)
    throws UnknownHostException{
    super(Message.FLOWREQUEST_TYPE);
    this.bytes = bytes;
    int pointer=0;
    try{
      source_interface = new IPv6Address(Array.
        getSubArray(bytes,pointer,IPv6Address.length));
      pointer += IPv6Address.length;
      destination_interface = new IPv6Address(Array.
        getSubArray(bytes,pointer,pointer+IPv6Address.length));
      pointer += IPv6Address.length;
      service_level = bytes[pointer++];
      time_stamp = PrimitiveConversions.getLong(
        Array.getSubArray(bytes,pointer, pointer+8));
      pointer += 8;
      if (bytes.length == INTSERV_SIZE) {
        requested_delay = PrimitiveConversions.getInt(
          Array.getSubArray(bytes,pointer, pointer+4));
        pointer += 4;
```

```
      requested_loss_rate = PrimitiveConversions.getInt(
        Array.getSubArray(bytes,pointer, pointer+4));
      pointer += 4;
      requested_throughput = PrimitiveConversions.getInt(
        Array.getSubArray(bytes,pointer, pointer+4));
    }
    else {
      user_id = PrimitiveConversions.getInt(
        Array.getSubArray(bytes,pointer, pointer+4));
      pointer += 4;
      byte DSCP = bytes[pointer++];
      int profile = PrimitiveConversions.getInt(
        Array.getSubArray(bytes,pointer, pointer+4));
      pointer += 4;
      byte scope = bytes[pointer++];
      byte action = bytes[pointer++];
      if (action == SLS.REMARK) {
        sls = new SLS(DSCP, profile, scope,
          action, bytes[pointer]);
      }
      else if (action == SLS.SHAPE) {
        sls = new SLS(DSCP, profile, scope,
          action, PrimitiveConversions.getInt(
          Array.getSubArray(bytes,pointer, pointer+4)));
      }
      else {
        sls = new SLS(DSCP, profile, scope, action);
      }
    }
  }
  catch(UnknownHostException uhe){
    throw new UnknownHostException(uhe.toString());
  }
}

/**
 * Convert this IntServ FlowRequest to its byte array form
 * using the parameters supplied.
 * @param source_interface The IPv6Address of the source.
 * @param destination_interface The IPv6Address of the
 * destination.
 * @param time_stamp The 8 byte time stamp.
 * @param requested_delay The maximum delay reuested.
 * @param requested_loss_rate The maximum loss rate requested.
 * @param requested_throughput The maximum throughput requested.
 */
private void convertToBytes (IPv6Address source_interface,
  IPv6Address destination_interface,
  byte service_level,
  long time_stamp,
  int requested_delay,
  int requested_loss_rate,
  int requested_throughput){

   //build the byte array
  bytes = Array.concat(source_interface.getAddress(),
    destination_interface.getAddress());
```

```
    bytes = Array.concat(bytes, service_level);
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(time_stamp));
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(requested_delay));
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(requested_loss_rate));
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(requested_throughput));
  }

  /**
   * Convert this DiffServ FlowRequest to its byte array form
   * using the parameters supplied.
   * @param source_interface The IPv6Address of the source.
   * @param destination_interface The IPv6Address of the
   * destination.
   * @param time_stamp The 8 byte time stamp.
   * @param user_id The user identification number.
   * @param sls The type of SLS requested for.
   */
  private void convertToBytes (IPv6Address source_interface,
    IPv6Address destination_interface,
    byte service_level,
    long time_stamp,
    int user_id,
    SLS sls){

     //build the byte array
    bytes = Array.concat(source_interface.getAddress(),
      destination_interface.getAddress());
    bytes = Array.concat(bytes, service_level);
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(time_stamp));
    bytes = Array.concat(bytes, //user_id.getBytes());
      PrimitiveConversions.getBytes(user_id));
    bytes = Array.concat(bytes, sls.getSLSBytes());
  }

  /**
   * Returns the IPv6Address of the source.
   * @return The IPv6Address of the source.
   */
  public IPv6Address getSourceInterface(){
    return source_interface;
  }

  /**
   * Returns the network address associated with the source
   * IPv6Address
   * @return The network address associated with the source
   * IPv6Address
   */
  public IPv6Address getSourceLink(){
    return source_interface.getNetworkAddress();
  }
```

```
/**
 * Returns the IPv6Address of the destination.
 * @return The IPv6Address of the destination.
 */
public IPv6Address getDestinationInterface(){
  return destination_interface;
}

/**
 * Returns the network address associated with the
 * destination IPv6Address
 * @return The network address associated with the
 * destination IPv6Address
 */
public IPv6Address getDestinationLink(){
  return destination_interface.getNetworkAddress();
}

/**
 * Returns the 8 byte time stamp associated with this Message.
 * @return The 8 byte time stamp associated with this Message.
 */
public long getTimeStamp(){
  return time_stamp;
}

/**
 * Returns the 1 byte service_level associated with this Message.
 * @return The 1 byte service_level associated with this Message.
 */
public byte getServiceLevel(){
  return service_level;
}

/**
 * Returns the 4 byte user_id associated with this Message.
 * @return The 4 byte user_id associated with this Message.
 */
public int getUser(){
  return user_id;
}

/**
 * Returns the requested delay associated with this Message.
 * @return The requested delay associated with this Message.
 */
public int getRequestedDelay(){
  return requested_delay;
}

/**
 * Returns the requested loss rate associated with this Message.
 * @return The requested loss rate associated with this Message.
 */
public int getRequestedLossRate(){
  return requested_loss_rate;
}
```

```
  /**
   * Returns the requested throughput associated with this Message.
   * @return The requested throughput associated with this Message.
   */
  public int getRequestedThroughput(){
    return requested_throughput;
  }

  /**
   * Returns The byte array representation of this Message.
   * @return The byte array representation of this Message.
   */
  public byte[] getBytes(){
    return bytes;
  }

  /**
   * Returns the length of this Message.
   * @return The length of this Message.
   */
  public short length(){
    try{
      return (short)bytes.length;
    }catch(NullPointerException npe){
      return 0;
    }
  }

  /**
   * Returns a <code>String</code> representation of this Message.
   * @return The <code>String</code> representation of this Message
   */
  public String toString(){
    String flow_request = "Source: " + source_interface.toString() +
      ",\n\t Destination: " + destination_interface.toString() +
      ",\n\t TS: " + time_stamp + ", Service Level; " + service_level
        + ", D: " + requested_delay + ", LR: " +
      requested_loss_rate + ", T: " + requested_throughput;
    //it is a Differentiated Service
    if (service_level == Server.DS_SERVICELEVEL) {
      flow_request = flow_request+", "+sls.toString();
    }
    return flow_request;
  }


}//end of FlowRequest class
```

APPENDIX E – SAAM MESSAGE.FLOWRESPONSE CLASS CODE

```
//12Dec1999[Henry]                - Modified (a lot have changed)
//11Dec1999[Dean, John or Cary]   - Created

package saam.message;

import java.net.UnknownHostException;
import saam.net.*;
import saam.util.*;

import saam.server.diffserv.*;
import saam.server.*;


/**
 * A Response to a flow request simply contains the timestamp
 * that was sent with the corresponding FlowRequest, and
 * the new flow id that has been assigned to the Object
 * requesting the flow.
 */
public class FlowResponse extends Message{

  //add by Henry for possible status of flow response
  public static final byte SERVICE_UNKNOWN = 0;
  public static final byte IS_ACCEPTED = 1;
  public static final byte DS_ACCEPTED = 2;
  public static final byte REJECTED = 3;
  public static final byte NEGOTIATED = 4;
  public static final byte UNREACHEABLE = 5;
  public static final byte SLA_NOT_AVAILABLE = 6;

  /** The byte length of an IntServ flow response  */
  public static final int INTSERV_SIZE = 13;  //8+1+4

  /**
   *  Message format:
   *    1                  8-11 / 3-17
   *  Result  Service_Level_Spec / Flow_Id
   */

  //added by Henry for result field of flow response
  private byte result = SERVICE_UNKNOWN;

  /** The flow_id assigned for the flow. */
      //will be truncated to 3 bytes by IPv6Header
  private int flow_id = Server.FLOWNUNREACHEABLE;

  /** The average delay negotiated  */
  private int delay = 0;
  /** The average rate of packet loss negotiated. */
  private int loss_rate = 0;
  /** The rate of data negotiated. */
  private int throughput = 0;
```

```
/** The service level spec for the flow. */
private SLS sls;

/** The hashcode that represent the user of this SLS */
private int user_id = 0;

/** The byte array which stores the message parameters  */
private byte[] bytes;

/** The time when this message is created  */
private long time_stamp;

/**
 * No-args constructor used by the server.
 */
public FlowResponse(){
  super(Message.FLOWRESPONSE_TYPE);
}

/**
 * Constructs a FlowResponse with the parameters supplied.
 * @param time_stamp The 8 byte time stamp associated with
 * this Message.
 * @param flow_ID The flow associated with this Message.
 */
public FlowResponse(long time_stamp, int flow_id){

  super(Message.FLOWRESPONSE_TYPE);
  this.time_stamp = time_stamp;
  this.flow_id = flow_id;

  bytes = Array.concat(
    PrimitiveConversions.getBytes(time_stamp),
    PrimitiveConversions.getBytes(flow_id));
}

/**For DiffServ
 * Constructs a DiffServ FlowResponse with the parameters
 * supplied (used when result == SERVICE_UNKNOWN
 * /REJECTED/UNREACHEABLE/SLA_NOT_AVAILABLE).
 * @param time_stamp The 8 byte time stamp associated with
 * this Message.
 * @param result The result associated with this Message.
 */
public FlowResponse(long time_stamp, byte result){
  this(time_stamp, result, 0);
}

/**
 * Constructs a FlowResponse with the parameters supplied
 * (used when result == IS_ACCEPTED).
 * @param time_stamp The 8 byte time stamp associated with
 * this Message.
 * @param flow_id The flow associated with this Message.
 * @param result The result associated with this Message.
 */
public FlowResponse(long time_stamp, byte result,
```

```
                        int flow_id){

    super(Message.FLOWRESPONSE_TYPE);
    this.time_stamp = time_stamp;
    this.result = result;
    this.flow_id = flow_id;

    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(time_stamp));
    bytes = Array.concat(bytes, result);
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(flow_id));
  }

  /**
   * Constructs a FlowResponse with the parameters supplied.
   * for QoS negotiation (used when reult == NEGOTIATED.
   * @param time_stamp The 8 byte time stamp associated with
   * this Message.
   * @param flow_id The flow associated with this Message.
   * @param result The result associated with this Message.
   * @param delay The maximum delay negotiable.
   * @param loss_rate The maximum loss rate negotiable.
   * @param throughput The maximum throughput negotiable.
   */
  public FlowResponse(long time_stamp, byte result,
         int delay, int loss_rate, int throughput){

    super(Message.FLOWRESPONSE_TYPE);
    this.time_stamp = time_stamp;
    this.result = result;
    this.delay = delay;
    this.loss_rate = loss_rate;
    this.throughput = throughput;

    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(time_stamp));
    bytes = Array.concat(bytes, result);
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(delay));
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(loss_rate));
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(throughput));
  }

  /**
   * Constructs a FlowResponse with the parameters supplied.
   * @param time_stamp The 8 byte time stamp associated with
   * this Message.
   * @param result The result associated with this Message.
   * @param user_id The user identification number.
   * @param sls The type of SLS requested for.
   */
  public FlowResponse(long time_stamp, byte result,
                      int user_id, SLS sls){
```

```
    super(Message.FLOWRESPONSE_TYPE);
    this.time_stamp = time_stamp;
    this.result = result;
    this.sls = sls;
    this.user_id = user_id;
    this.flow_id = 0;

    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(time_stamp));
    bytes = Array.concat(bytes,result);
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(user_id));
    bytes = Array.concat(bytes,sls.getDSCP());
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(sls.getProfile()));
    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(sls.getScope()));
    byte action = sls.getDispositionAction();
    bytes = Array.concat(bytes,action);
    if (action == SLS.REMARK) {
      bytes = Array.concat(bytes,sls.getActionByte());
    }
    else if (action == SLS.SHAPE) {
      bytes = Array.concat(bytes,
        PrimitiveConversions.getBytes(sls.getActionInt()));
    }
  }

  /**
   * Construct this Message from a byte array that is presumed
   * to conform to the proper format for this Message.  Presumably,
   * this constructor is called when the receiving PacketFactory
   * gets the byte array that represents this Message - a byte
   * array that was presumably generated when the sender of this
   * Message called the getBytes() method after creating this
   * Message and before sending it.
   */
  public FlowResponse(byte[] bytes)
    throws UnknownHostException{
    super(Message.FLOWRESPONSE_TYPE);
    this.bytes = bytes;
    int pointer=0;
    time_stamp = PrimitiveConversions.getLong(
      Array.getSubArray(bytes,pointer, 8));
    pointer = pointer + 8;
    //System.out.println(time_stamp+"; "+pointer);
    //added by Henry
    result = bytes[pointer++];
    //System.out.println(result+"; "+pointer);
    //System.out.println(bytes.length);
    if (bytes.length <= INTSERV_SIZE) {
    //System.out.println("Retreving flow_id");
          flow_id = PrimitiveConversions.getInt(
            //Array.getSubArray(bytes, pointer, bytes.length));
            Array.getSubArray(bytes,pointer, pointer+4));
    }
    else {
```

```
    user_id = PrimitiveConversions.getInt(
      Array.getSubArray(bytes,pointer, pointer+4));
    pointer += 4;
    byte DSCP = bytes[pointer++];
    int profile = PrimitiveConversions.getInt(
      Array.getSubArray(bytes,pointer, pointer+4));
    pointer += 4;
    byte scope = bytes[pointer++];
    byte action = bytes[pointer++];
    if (action == SLS.REMARK) {
      sls = new SLS(DSCP, profile, scope,
                action, bytes[pointer]);
    }
    else if (action == SLS.SHAPE) {
      sls = new SLS(DSCP, profile, scope, action,
          PrimitiveConversions.getInt(
          Array.getSubArray(bytes,pointer, pointer+4)));
    }
    else {
      sls = new SLS(DSCP, profile, scope, action);
    }
  }
}

/**
 * Returns the 8 byte time stamp associated with this Message.
 * @return The 8 byte time stamp associated with this Message.
 */
public long getTimeStamp(){
  return time_stamp;
}

/**
 * Returns the flow ID associated with this event.
 * @return The flow ID associated with this event.
 */
public int getFlowId(){
  return flow_id;
}

/** added by Henry
 * Returns the result associated with this event.
 * @return The result associated with this event.
 */
public byte getResult(){
  return result;
}

/** added by Henry
 * Returns the result associated with this event.
 * @return The result associated with this event.
 */
public int getUserId(){
  return user_id;
}
```

```
  /**
   * Returns The byte array representation of this Message.
   * @return The byte array representation of this Message.
   */
  public byte[] getBytes(){
    return bytes;
  }

  /**
   * Returns the length of this Message.
   * @return The length of this Message.
   */
  public short length(){
    try{
      return (short)bytes.length;
    }catch(NullPointerException npe){
      return 0;
    }
  }

  /**
   * Returns a <code>String</code> representation of this Message.
   * @return The <code>String</code> representation of this Message
   */
  public String toString(){

    String flow_response = "This flow response message contains: "
                +", Result = "+result+", Time Stamp = "+time_stamp;
    if (flow_id != 0) {  //is it an Integrated Service
      flow_response = flow_response+", Flow_ID = "+flow_id;
    }
    else if (sls != null) {  //it is a Differentiated Service
      flow_response = flow_response+", ServiceLevelSpec = "
                  +sls.toString();
    }
    return flow_response;
  }

}//end of FlowResponse class
```

APPENDIX F – SAAM.MESSAGE.FLOWTERMINATION CLASS CODE

```
//14Feb2000[Henry]  - Created,

package saam.message;

import java.net.UnknownHostException;
import saam.net.*;
import saam.util.*;
import saam.server.diffserv.*;


/**
 * A FlowTermination is sent by the router to the server to
 * inform the server that a flow will no longer be used..
 */
public class FlowTermination extends Message{

  /**
   *  Message format:
   *        3
   *  Flow_Id
   */

  /** The flow_id assigned for the flow. */
  private int flow_id = 0;  //will be truncated to 3 bytes
                            //by PacketFactory

  /** The service level spec for the flow. */
  private SLS sls;

  /** The byte array which stores the message parameters  */
      private byte[] bytes;

  /**
   * No-args constructor used by the server.
   */
  public FlowTermination(){
    super(Message.FLOWTERMINATION_TYPE);
  }

  /**
   * Constructs a FlowTermination with the parameters supplied.
   * @param time_stamp The 8 byte time stamp associated with
   * this Message.
   * @param flow_ID The flow associated with this Message.
   */
  public FlowTermination(int flow_id){

    super(Message.FLOWTERMINATION_TYPE);
    this.flow_id = flow_id;

    bytes = Array.concat(bytes,
      PrimitiveConversions.getBytes(flow_id));
  }
```

```
  /**
   * Construct this Message from a byte array that is presumed
   * to conform to the proper format for this Message. Presumably,
   * this constructor is called when the receiving PacketFactory
   * gets the byte array that represents this Message - a byte
   * array that was presumably generated when the sender of this
   * Message called the getBytes() method after creating
   * this Message and before sending it.
   */
  public FlowTermination(byte[] bytes)
    throws UnknownHostException{
    this.bytes = bytes;
    int pointer=0;
          flow_id = PrimitiveConversions.getInt(
            Array.getSubArray(bytes,pointer, pointer+4));
  }

  /**
   * Returns the flow ID associated with this event.
   * @return The flow ID associated with this event.
   */
  public int getFlowId(){
    return flow_id;
  }

  /**
   * Returns The byte array representation of this Message.
   * @return The byte array representation of this Message.
   */
  public byte[] getBytes(){
    return bytes;
  }

  /**
   * Returns the length of this Message.
   * @return The length of this Message.
   */
  public short length(){
    try{
      return (short)bytes.length;
    }catch(NullPointerException npe){
      return 0;
    }
  }

  /**
   * Returns a <code>String</code> representation of this Message.
   * @return The <code>String</code> representation of this Message
   */
  public String toString(){
    String flow_termination =
      "This flow termination message contains: Flow_ID = "+flow_id;
    return flow_termination;
  }

}//end of FlowTerminatin class
```

APPENDIX G – SAAM.MESSAGE.SLSTABLEENTRY CLASS CODE

```
// 8Feb2000[Henry]            - Modified
// 10Jan2000[Henry]           - Created

package saam.message;

import saam.server.diffserv.*;
import saam.util.*;

/**
 * A SLSTableEntry Message that contains the SLS that will
 * be sent to the router to update its SLSTable.
 */
public class SLSTableEntry extends Message{

  /** The service level spec for the flow. */
  private SLS sls;

  /** The integer value that uniquely identifies the user
  who owns this SLS */
  private int user_id = 0;

  /** The integer value that uniquely identifies the node
  who owns this SLS */
  private int node_id = 0;

  /** The byte array which stores the message parameters  */
  private byte[] bytes;

  /** The byte length of a SLSTableEnry which has a SLS
  that is to be removed from the SLSTable  */
  public static final int REMOVE_SLS_TYPE = 8;

  /**
   * Constructs a SLSTableEntry with the parameters supplied.
   * @param   sls  The SLS to be contained in this message
   */
  public SLSTableEntry(int user_id, int node_id){
    super(Message.SLSTABLEENTRY_TYPE);
    this.user_id = user_id;
    this.node_id = node_id;

    bytes = Array.concat(PrimitiveConversions.getBytes(user_id),
            PrimitiveConversions.getBytes(node_id));
  }

  /**
   * Constructs a SLSTableEntry with the parameters supplied.
   * @param   sls  The SLS to be contained in this message
   */
  public SLSTableEntry(int user_id, SLS sls){
    super(Message.SLSTABLEENTRY_TYPE);
    this.user_id = user_id;
    this.sls = sls;
```

```
  bytes = Array.concat(PrimitiveConversions.getBytes(user_id),

    sls.getSLSBytes());
}

/**
 * Construct this Message from a byte array that is presumed
 * to conform to the proper format for this Message. Presumably,
 * this constructor is called when the receiving PacketFactory
 * gets the byte array that represents this Message - a byte
 * array that was presumably generated when the sender of this
 * Message called the getBytes() method after creating
 * this Message and before sending it.
 */
public SLSTableEntry(byte[] bytes) {
  this.bytes = bytes;
  int pointer=0;
  user_id = PrimitiveConversions.getInt(
    Array.getSubArray(bytes,pointer, pointer+4));
  pointer += 4;
  if (bytes.length == REMOVE_SLS_TYPE) {
    node_id = PrimitiveConversions.getInt(
    Array.getSubArray(bytes,pointer, pointer+4));
        pointer += 4;
  }
  else {
    byte DSCP = bytes[pointer++];
    int profile = PrimitiveConversions.getInt(
          Array.getSubArray(bytes,pointer, pointer+4));
    pointer += 4;
    byte scope = bytes[pointer++];
    byte action = bytes[pointer++];
    if (action == SLS.REMARK) {
      sls = new SLS(DSCP, profile, scope,
        action, bytes[pointer]);
    }
    else if (action == SLS.SHAPE) {
      sls = new SLS(DSCP, profile, scope, action,
        PrimitiveConversions.getInt(
        Array.getSubArray(bytes,pointer, pointer+4)));
    }
    else {
      sls = new SLS(DSCP, profile, scope, action);
    }
  }
}

/**
 * Returns the user_id stored in this Message
 * @return The user_id stored in this Message
 */
public int getUserId(){
  return user_id;
}
```

```
  /**
   * Returns the user_id stored in this Message
   * @return The user_id stored in this Message
   */
  public int getNodeId(){
    return node_id;
  }

  /**
   * Returns the SLS stored in this Message
   * @return The sls stored in this Message
   */
  public SLS getSLS(){
    return sls;
  }

  /**
   * Returns the byte array representation of this Message.
   * @return The byte array representation of this Message.
   */
  public byte[] getBytes(){
    return bytes;
  }

  /**
   * Returns the length of this Message.
   * @return The length of this Message.
   */
  public short length(){
    try{
      return (short)bytes.length;
    }catch(NullPointerException npe){
      return 0;
    }
  }

  /**
   * Returns a <code>String</code> representation of this Message.
   * @return The <code>String</code> representation of this Message
   */
  public String toString(){
    String slsMessage;
    if (node_id == 0) {
      slsMessage = "This SLSTableEntry message contains: "
        + "UserId = " + user_id + sls.toString();
    }
    else {
      slsMessage = "This SLSTableEntry message contains: "
        + "Node_id = " +node_id;
    }
    return slsMessage;
  }

}//end of SLSTableEntry class
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H – SAAM.MESSAGE.MESSAGE CLASS CODE

```
//10Jan2000[Henry]  - Added declaration for SERVICELEVELSPEC_TYPE
//13Dec99[Henry]  - Added declaration for FLOWRESPONSE_TYPE and
RESOURCEALLOCATION_TYPE
//08Dec99[Efain]  - Added declarations
//01Aug99[Dean]   -Created..

package saam.message;


/**
 * The Message class provides a convenient way for Objects to
communicate
 * with one another over a SAAM network.  The standard JDK does not
currently
 * provide a means to serialize objects over UDP.  This class does just
that.
 * Subclasses need to be written as follows to enable this
functionality:
 *
 * 1. Provide a constructor that accepts a byte array as its only
parameter.
 * 2. Override the getBytes method in such a way that it returns a byte
array
 *    that contains the values of the variables to be transferred.
 * 3. Ensure that the constructor mentioned above is set up to properly
 *    parse the byte array and rebuild the variables as they were
originally.
 * 4. Ensure that the length method returns the actual length of the
byte array.
 */

public abstract class Message{

  //for default type to support old version
  public static final byte MESSAGE_DEFAULT_TYPE    = 1;

  //for fault tolerance
  public static final byte HEARTBEAT_QUERY_TYPE    = 2;
  public static final byte HEARTBEAT_RESPONSE_TYPE = 3;

  //for control channel construction
  public static final byte UCM_TYPE                = 4;
  public static final byte DCM_TYPE                = 5;
  public static final byte PARENT_NOTIFICATION_TYPE= 6;
  public static final byte RESERVED1_TYPE          = 7;

  //following types reserved for flow reservation
  public static final byte FLOWRESPONSE_TYPE       = 8;
  public static final byte FLOWREQUEST_TYPE        = 9;
  public static final byte RESERVED4_TYPE          = 10;
  public static final byte RESERVED5_TYPE          = 11;
  public static final byte FLOWTERMINATION_TYPE    = 12;
```

```
  //following types reserved for probing
  public static final byte RESERVED7_TYPE          = 13;
  public static final byte RESERVED8_TYPE          = 14;
  public static final byte RESERVED9_TYPE          = 15;

  //following types reserved for resource manegememnt
  public static final byte RESOURCEALLOCATION_TYPE = 16;
  public static final byte SLSTABLEENTRY_TYPE      = 17;
  public static final byte RESERVED12_TYPE         = 18;

  //following types are reserved for security
  public static final byte RESERVED13_TYPE         = 19;
  public static final byte RESERVED14_TYPE         = 20;
  public static final byte RESERVED15_TYPE         = 21;
  public static final byte RESERVED16_TYPE         = 22;


  /**
   * type is a byte value to represent different type of messages
   */

  protected byte type;


  /**
   * No-args constructor initializes the type to a default value which
is 1.
   * @param none
   */

  public Message(){

    type =  MESSAGE_DEFAULT_TYPE ;

  }//end Message()


  /**
   * Constructs a Messagee with the supplied type_id parameter.
   * @param type_id byte value representing different types of messages
   */

  public Message(byte type_id){

    this.type = type_id;

  }//end Message()


  /**
   * Returns the type value.
   * @return byte the type value.
   */

  public byte getType(){

    return type;
```

```
  }//end getType()


  /**
   * Sets the type value to the parameter given.
   * @param type_id byte value which represents the message type.
   * @return void
   */

  public void setType(byte type_id){

    type = type_id;

  }//end setType()

  /**
   * Abstract method. Returns the length of this Message.
   * @param none
   * @return short the length of this Message.
   */

  public abstract short length();

  /**
   * Abstract method. Returns The byte array representation of this
Message.
   * @param none
   * @return byte[] the byte array representation of this Message.
   */

  public abstract byte[] getBytes();

  /**
   * Returns a String representation of this Message.
   * @param none
   * @return String the String representation of this Message
   */

  public String toString(){

    return "Message";

  }//end toString()

}//end class Message

//end Message.java
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I – SAAM SERVER.CLASSOBJECTSTRUCTURE CLASS CODE

```
//23Feb2000[Henry]    -  modified
// 13Dec99 [Henry]    -  Changed setTargetThroughput to
setAllocatedThroughput
//                       Added getPathsThatSupportFlowRequest
// 09Dec99 [Xie]      -  Modified getNewFlowID()

package saam.server;

import saam.net.*;
import saam.message.*;
import saam.util.*;
import java.net.*;
import java.sql.*;
import java.util.*;
import java.io.*;

/**
 * The <em>ClassObjectStructure</em> is a Path Information Base object
within the
 * SAAM architecture that performs operations on class objects
containing the
 * information needed to obtain a picture of the network for use in
assigning
 * flows to paths.
 */
public class ClassObjectStructure extends PathInformationBase{

/** Contains all of the known router nodes. */
Hashtable nodes;
/** Contains all of the known router interfaces. */
Hashtable interfaces;
/** Contains service level pipes. */
Vector slps;
/** Describes the QoS parameters for a service level pipe. */
SLP_QoS slp_qos;
/** Contains all of the known links. */
Hashtable links;
/** Contains all of the constructed paths. */
Hashtable paths;
/** Describes the characteristics of a path. */
Path path;
/** Contains all of the assigned flows. */
Hashtable flows;
/** Describes the QoS characteristics of an assigned flow. */
Flow_QoS flow_qos;
/** Contains a sequence of service level pipes. */
Vector SLP_Sequence;
/** Describes a service level pipe. */
ServiceLevelPipe slp;
/** A boolean that will allow the showing of comments. */
private boolean showComments = false;
private SAAMRouterGui gui;
```

```
public static final int MIN_APP_FLOW_ID = 65;
public static final int MAX_FLOW_ID = 16777215; // 2^24 - 1
protected int newAppFlowID;

private static final int WSPath = 0;      //Widest-Shortest Path
private static final int SWPath = 1;      //Shortest-Widest Path

//public static float[] throughputForSL;
private static float[] loadingfactor;
private static float[] increasingfactor;
private static float[] borrowingfactor;

  /**
   * The <em>SLP_QoS</em> defines the QoS charateristics of a service
level pipe.
   */
  private class SLP_QoS {
    /** The maximum delay expected on this SLP. */
    int targetDelay=0;
    /** The maximum loss rate expected on this SLP. */
    int targetLossRate=0;
    /** The amount of bandwidth that this SLP should be able to
provide. */
    int targetThroughput=0;
    /** The amount of delay being observed at this SLP. */
    int observedDelay=0;
    /** The loss rate being observed at this SLP. */
    int observedLossRate=0;
    /** The utilization being observed at this SLP. */
    int observedUtilization=0;
    /** The service level of this SLP. */
    int serviceLevel=0;  //Added by Henry
    public String toString(){
      return "SLP_QoS:target D="+targetDelay+", LR="+targetLossRate+",
T="
        +targetThroughput+", observed D="+observedDelay+",
LR="+observedLossRate
        +", U="+observedUtilization+", SL="+serviceLevel;
    }
  }

  /**
   * The <em>Path</em> defines the characteristics of a path.
   */
  private class Path {
    /** The first router in the path. */
    int sourceRouter=0;
    /** The last router in the path. */
    int destinationRouter=0;
    /** The total delay a flows traversing this path experiences. */
    int effectiveDelay=0;
    /** The totoal loss rate a flow traversing this path experiences.
*/
    int effectiveLossRate=0;
    /** The amount on bandwidth still available on this path. */
    int effectiveThroughputRemaining=0;
    /** The flows that are assigned to this path. */
```

```
    Hashtable flows = new Hashtable();
    /** The sequence of service level pipes that make up this path. */
    Vector SLPSequence = new Vector();
    public String toString(){
      return "Path: from "+sourceRouter+" to "+destinationRouter+" with
flows:"
        +flows;
    }
  }

  /**
   * The <em>Flow_QoS</em> defines the QoS characteristics of a flow.
   */
  private class Flow_QoS {
    /** The maximum amount of delay that the flow is expected to
experience. */
    int negotiatedDelay=0;
    /** The maximum loss rate that the flow is expected to experience.
*/
    int negotiatedLossRate=0;
    /** The maximum amount of bandwidth that a flow is expected to
consume. */
    int negotiatedThroughput=0;
    /** The average delay experienced by a flow. */
    int observedDelay=0;
    /** The loss rate experienced by a flow. */
    int observedLossRate=0;
    /** The amount of bandwdith being consumed by a flow. */
    int observedThroughput=0;
  }

  /**
   * The <em>ServiceLevelPipe</em> defines characteristics of a service
level
   * pipe.
   */
  private class ServiceLevelPipe {
    /** The IPv6 address of the interface. */
    IPv6Address address;
    /** The level of service that this SLP expects to provide to flows.
*/
    int serviceLevel=0;
  }


//*****************************************************************
********
// These methods are used to initialize the path information base.
//*****************************************************************
*******/

  /**
   * Constructs a ClassObjectStructure object that will be used to
manipulate
   * the class objects of this path information base.
   */
  public ClassObjectStructure(){
```

```
    gui = new SAAMRouterGui("PIB");
    newAppFlowID = MIN_APP_FLOW_ID;
    //if (showComments){
      gui.sendText("PIB: ClassObjectStructure: Constructor executed.");
    //}
    setLoadingFactor();
    //setIncreasingFactor();
    //setBorrowingFactor();
  }

  //Henry
  public ClassObjectStructure(SAAMRouterGui gui){
    this.gui = gui;
    newAppFlowID = MIN_APP_FLOW_ID;
    //if (showComments){
      gui.sendText("PIB: ClassObjectStructure: Constructor executed.");
    //}
    setLoadingFactor();
    //setIncreasingFactor();
    //setBorrowingFactor();
  }

  public void setLoadingFactor() {
    loadingfactor = new float[Server.NUMBEROFSERVICELEVELS];
    loadingfactor[Server.CONTROL_SERVICELEVEL] = 1f;  //Control Packets
    loadingfactor[Server.IS_SERVICELEVEL] = 0.7f;     //INTSERV
    loadingfactor[Server.DS_SERVICELEVEL] = 0.9f;     //DIFFSERV
    loadingfactor[Server.BE_SERVICELEVEL] = 1f;       //BESTEFFORT
    loadingfactor[Server.OTHER_SERVICELEVEL] = 0f;    //Tag Packets
  }
/*  public void setIncreasingFactor() {
    increasingfactor = new float[Server.NUMBEROFSERVICELEVELS];
    increasingfactor[Server.CONTROL_SERVICELEVEL] = 0f;  //Control
Packets
    increasingfactor[Server.IS_SERVICELEVEL] = 0.1f;     //INTSERV
    increasingfactor[Server.DS_SERVICELEVEL] = 0.1f;     //DIFFSERV
    increasingfactor[Server.BE_SERVICELEVEL] = 0f;       //BESTEFFORT
    increasingfactor[Server.OTHER_SERVICELEVEL] = 0f;    //Tag Packets
  }
  public void setBorrowingFactor() {
    borrowingfactor = new float[Server.NUMBEROFSERVICELEVELS];
    borrowingfactor[Server.CONTROL_SERVICELEVEL] = 0f;  //Control
Packets
    borrowingfactor[Server.IS_SERVICELEVEL] = 0.1f;     //INTSERV
    borrowingfactor[Server.DS_SERVICELEVEL] = 0.1f;     //DIFFSERV
    borrowingfactor[Server.BE_SERVICELEVEL] = 0f;       //BESTEFFORT
    borrowingfactor[Server.OTHER_SERVICELEVEL] = 0f;    //Tag Packets
  }*/

  /**
   * Removes all current path data from the database.. Its most
commonly used
   * during initialization of a SAAM server for a new network.
   */
  public void deleteAllData() {
    nodes = new Hashtable();
    links = new Hashtable();
```

```
    paths = new Hashtable();
    interfaces = new Hashtable();
    if (showComments){
      gui.sendText("PIB: deleteAllData: All data deleted.");
    }
    initializeAllocation();
  }

  /**
   * Removes all current path data from the database.. Its most
commonly used
   * during initialization of a SAAM server for a new network.
   */
  public void initializeAllocation() {
  }


//*********************************************************************
********
// These methods are used to process link state advertisements from
routers
//*********************************************************************
*******/

  /**
   * Determines whether a given router exists yet within the PIB.
   * @param IPv6Addresses A vector of interface addresses contained
within
   * a Hello or an LSA message.
   * @returns node_id The id of the node containing at least one of the
   * interface addresses in the vector that was passed.
   */
  public int doesRouterExist(Vector IPv6Addresses){
    Integer myNodeId;
    int node_id = 0;
    IPv6Address myIPv6Address;
    // for each of the interface IPv6 addresses that were passed in
    for (int i = 0; i < IPv6Addresses.size(); i++) {
      myIPv6Address = (IPv6Address)IPv6Addresses.elementAt(i);
      Enumeration e = nodes.keys();
      // for each of the node ids in the PIB
      while(e.hasMoreElements()){
        myNodeId = (Integer)e.nextElement();
        Hashtable myNode = (Hashtable)nodes.get(myNodeId);
        // if any of its address equals the address that was passed in
        if (myNode.containsKey(myIPv6Address.toString())){
          node_id = myNodeId.intValue();
        }
      }
    }
    if (showComments){
      if (node_id != 0)
        gui.sendText("PIB: doesRouterExist: Router " +node_id+ "
exists.");
      else
        gui.sendText("PIB: doesRouterExist: Router is not in
database.");
```

```
    }
    return node_id;
  }

  /**
   * Finds an unassigned node id and adds it to the PIB. It is commonly
used
   * for assigning a new node_id to a previously unknown router.
   * @returns max_node_id An unassigned router id.
   */
  public int getNewNodeId(){
    int max_node_id = 0;
    Enumeration e = nodes.keys();
    // for each of the node ids in the PIB
    while(e.hasMoreElements()){
      Integer myNodeId = (Integer)e.nextElement();
      // if the id is greater than the max
      if (max_node_id < myNodeId.intValue())
        // then assign it as the max
        max_node_id = myNodeId.intValue();
    }
    // increment the max to get a new max
    max_node_id++;
    // enter this node id into the PIB
    nodes.put(new Integer(max_node_id), new Hashtable());
    if (showComments){
      gui.sendText("PIB: assignNewNodeId: Router's id assigned: "
        + max_node_id);
    }
    return max_node_id;
  }

  /**
   * Determines whether a given interface exists yet within the PIB.
   * @param myIPv6Address The interface address contained within a
hello or LSA
   * message.
   * @returns found True if the interface address already exists within
the PIB.
   */
  public boolean doesInterfaceExist(IPv6Address myIPv6Address){
    boolean found = false;
    Integer myNodeId;
    Enumeration e = nodes.keys();
    // for each node in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if any of its interface addresses equal LSA interface address
      if (myNode.containsKey(myIPv6Address.toString()))
        found = true;
    }
    if (showComments){
      if (found)
        gui.sendText("PIB: doesInterfaceExist: Interface "
          + myIPv6Address.toString() + " is found.");
      else
```

```
        gui.sendText("PIB: doesInterfaceExist: Interface "
          + myIPv6Address.toString() + " is not found.");
    }
    return found;
  }

  /**
   * Determines whether a given link exists yet within the PIB.
   * @param address The IPv6 address of an interface.
   * @returns found True if the link address already exists within the
PIB.
   */
  public boolean doesLinkExist(IPv6Address address){
    boolean found = false;
    // if the links known to the PIB contains this address
    if (links.containsKey(address.getNetworkAddress().toString()))
      found = true;
    if (showComments){
      if (found)
        gui.sendText("PIB: doesLinkExist: Link "
          + address.getNetworkAddress() + " is found.");
      else
        gui.sendText("PIB: doesLinkExist: Link "
          + address.getNetworkAddress() + " is not found.");
    }
    return found;
  }

  /**
   * Adds a new link to the PIB.
   * @param address The IPv6 address of an interface.
   * @param max_bandwidth The max transmission rate over this network
segment.
   */
  public void addLink(IPv6Address address, int max_bandwidth){
    // add the link entry to the hash table
    links.put(address.getNetworkAddress().toString(),new
Integer(max_bandwidth));
    if (showComments){
      gui.sendText("PIB: addLink: Link " + address.getNetworkAddress()
        + " is added.");
    }
  }

  /**
   * Adds a new interface to the PIB.
   * @param node_id The id of the router whose interface is being
added.
   * @param address The IPv6 address of an interface.
   */
  public void addInterface(int node_id, IPv6Address address){
    // get the node assigned to this node id
    Hashtable myNode = (Hashtable)nodes.get(new Integer(node_id));
    // add this new interface to the node
    myNode.put(address.toString(), new Vector());
    if (showComments){
```

```
      gui.sendText("PIB: addInterface: Interface "+ address +" is
added.");
    }
  }

  /**
   * Determines whether a service level pipe exists yet within the PIB.
   * @param address The IPv6 address of an interface.
   * @param service_level The level of service that this logical pipe
is
   * providing.
   * @returns found True if this SLP is already in the PIB.
   */
  public boolean doesSLPExist(IPv6Address myIPv6Address, int
service_level){
    boolean found = false;
    Integer myNodeId = new Integer(0);
    Enumeration e_node_ids = nodes.keys();
    // for each of the node ids in the PIB
    while(e_node_ids.hasMoreElements()){
      myNodeId = (Integer)e_node_ids.nextElement();
      Hashtable Interfaces = (Hashtable)nodes.get(myNodeId);
      // if this interface's address equals the interface address of
this slp
      if (Interfaces.containsKey(myIPv6Address.toString())){
        Vector slps = (Vector)Interfaces.get(myIPv6Address.toString());
        // if this interface has more service levels than this service
level
        if (slps.size() >= service_level) {
          found = true;
        }
      }
    }
    if (showComments){
      if (found)
        gui.sendText("PIB: doesSLPExist: SLP from router "
          + myNodeId + " is found.");
      else
        gui.sendText("PIB: doesSLPExist: SLP from router "
          + myNodeId + " is not found.");
    }
    return found;
  }

  /**
   * Updates the status of a known SLP's delay, loss_rate, and
throughput.
   * @param address The IPv6 address of an interface.
   * @param service_level The level of service that this logical pipe
is
   * providing.
   * @param delay The average delay experienced by a packet's stay in
the
   * particular SLP outbound queue.
   * @param loss_rate The average loss_rate experienced by packets in a
   * particular SLP.
```

```
   * @param throughput The average throughput provided by a particular
SLP.
   */
  public void updateSLP(IPv6Address address, int service_level, int
delay,
    int loss_rate, int throughput){
    Integer myNodeId = new Integer(0);
    Enumeration e = nodes.keys();
    // for each of the nodes in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if this node contains an interface with the address passed in
      if (myNode.containsKey(address.toString())){
        Vector myInterface = (Vector)myNode.get(address.toString());
        // if this interface has more service levels than this service
level
        if (myInterface.size() >= service_level) {
          slp_qos = (SLP_QoS)myInterface.elementAt(service_level);
          // update it with these new values
          slp_qos.observedDelay = delay;
          slp_qos.observedLossRate = loss_rate;
          slp_qos.observedUtilization = throughput;
          myInterface.setElementAt(slp_qos,service_level);
          //myInterface.insertElementAt(slp_qos,service_level);
          if (showComments){
            gui.sendText("PIB: updateSLP: SLP " + service_level
              + " is assigned delay="+delay+",loss_rate="+loss_rate
              + ",throughput="+throughput);
          }
        }
      }
    }
    if (showComments){
      gui.sendText("PIB: updateSLP: SLP " + service_level +" is
updated.");
    }
  }

  /**Henry
   * Updates the status of a known SLP's delay, loss_rate, and
throughput.
   * @param address The IPv6 address of an interface.
   * @param service_level The level of service that this logical pipe
is
   * providing.
   * @param delay The average delay experienced by a packet's stay in
the
   * particular SLP outbound queue.
   * @param loss_rate The average loss_rate experienced by packets in a
   * particular SLP.
   * @param throughput The average throughput provided by a particular
SLP.
   */
  public void updateSLP(IPv6Address address, byte service_level, int
delay,
    int loss_rate, int throughput){
```

```
    Integer myNodeId = new Integer(0);
    Enumeration e = nodes.keys();
    // for each of the nodes in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if this node contains an interface with the address passed in
      if (myNode.containsKey(address.toString())){
        Vector myInterface = (Vector)myNode.get(address.toString());
        // if this interface has more service levels than this service
level
        if (myInterface.size() >= service_level) {
          slp_qos = (SLP_QoS)myInterface.elementAt(service_level);
          // update it with these new values
          slp_qos.observedDelay = delay;
          slp_qos.observedLossRate = loss_rate;
          slp_qos.observedUtilization = throughput
/*slp_qos.targetThroughput*/
                                    + slp_qos.observedUtilization;
          myInterface.setElementAt(slp_qos,service_level);
          if (showComments){
            gui.sendText("PIB: updateSLP: SLP " + service_level
              + " is assigned delay="+delay+",loss_rate="+loss_rate
              + ",throughput="+throughput);
          }
        }
      }
    }
    if (showComments){
      gui.sendText("PIB: updateSLP: SLP " + service_level +" is
updated.");
    }
  }//end updateSLP

  /**
   * Updates the target attributes of a known SLP's delay, loss_rate,
   * and throughput.
   * @param address The IPv6 address of an interface.
   * @param service_level The level of service that this logical pipe
is
   * providing.
   * @param delay The targeted delay experienced by a packet's stay in
the
   * particular SLP outbound queue.
   * @param loss_rate The targeted loss_rate experienced by packets in
a
   * particular SLP.
   * @param throughput The targeted throughput provided by a particular
SLP.
   */
  public void updateSLPTarget(IPv6Address address,
    byte service_level, int delay, int loss_rate, int throughput){
    Integer myNodeId = new Integer(0);
    Enumeration e = nodes.keys();
    // for each of the nodes in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
```

```
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if this node contains an interface with the address passed in
      if (myNode.containsKey(address.toString())){
        Vector myInterface = (Vector)myNode.get(address.toString());
        //System.out.println("My interface = "+myInterface);
        // if this interface has more service levels than this service
level
        if (myInterface.size() > service_level) {
          slp_qos = (SLP_QoS)myInterface.elementAt(service_level);
          System.out.println("PIB: updateSLPTarget (before): SLP QoS =
" + slp_qos);
          // update it with these new values
          slp_qos.targetDelay = delay;
          slp_qos.targetLossRate = loss_rate;
          slp_qos.targetThroughput =
slp_qos.targetThroughput+throughput;
          System.out.println("PIB: updateSLPTarget (after): SLP QoS = "
+ slp_qos);
          myInterface.setElementAt(slp_qos,service_level);
          if (showComments){
            gui.sendText("PIB: updateSLPTarget: SLP " + service_level
            //System.out.println("PIB: updateSLPTarget: SLP " +
service_level
              + " is assigned delay="+delay+",loss_rate="+loss_rate
              + ",throughput="+throughput);
          }
        }
      }
    }
    if (showComments){
      gui.sendText("PIB: updateSLPTarget: SLP "
                + service_level +" is updated.");
    }
  }//end updateSLPTarget

  /**
   * Adds a previously unknown SLP to the PIB along with its targeted
QoS.
   * @param address The IPv6 address of an interface.
   * @param service_level The level of service that this logical pipe
is
   * providing.
   * @param target_delay The average delay experienced by a packet's
stay in the
   * particular SLP outbound queue.
   * @param target_loss_rate The average loss_rate experienced by
packets in a
   * particular SLP.
   * @param target_throughput The average throughput provided by a
particular SLP.
   */
  public void addSLP(IPv6Address address, int service_level, int
target_delay,
    int target_loss_rate, int target_throughput){
    int delay = 0, loss_rate = 0, throughput = 0;
    Integer myNodeId = new Integer(0);
    Enumeration e = nodes.keys();
```

```
    // for each node in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if any interface has this same IPv6 address
      if (myNode.containsKey(address.toString())){
        Vector myInterface = (Vector)myNode.get(address.toString());
        SLP_QoS slp_qos = new SLP_QoS();
        slp_qos.targetDelay = target_delay;
        slp_qos.targetLossRate = target_loss_rate;
        slp_qos.targetThroughput = target_throughput;
        slp_qos.observedDelay = delay;
        slp_qos.observedLossRate = loss_rate;
        slp_qos.observedUtilization = throughput;
        slp_qos.serviceLevel = service_level;   //Henry
        // add this new slp to this interface
        myInterface.insertElementAt(slp_qos,service_level);
      }
    }
    if (showComments){
    //  System.out.println("PIB: addSLP: SLP " + service_level + " is
added to "
      gui.sendText("PIB: addSLP: SLP " + service_level + " is added to
"
        +address);
    }
  }

  /**Henry
   * Gets the least bandwitdth possible for a path that traverse an
array of
   * interfaces
   * @param   address  The array of interface address
   * @param   service_level  The service level of this path
   * @return  The remaining throughput that may be allocated
   */
  public int getRemainingThroughput(IPv6Address[] address, byte
service_level){
    int remainingThroughput = 0;
    for (int i=0; i<address.length; i++) {
      int throughput =
getRemainingThroughput(address[i],service_level);
      if (remainingThroughput < throughput) {
        remainingThroughput = throughput;
      }
    }
    return remainingThroughput;
  }


  /**Henry
   * Gets the least bandwitdth possible for a path that traverse the
   * interface specified
   * @param   address  The interface address
   * @param   service_level  The service level of this path
   * @return  The remaining throughput that may be allocated
   */
```

```
  public int getRemainingThroughput(IPv6Address address, byte
service_level){
    //, int delay, int loss_rate, int throughput){
    int remainingThroughput = 0;
    Integer myNodeId = new Integer(0);
    Enumeration e = nodes.keys();
    // for each of the nodes in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if this node contains an interface with the address passed in
      if (myNode.containsKey(address.toString())){
        Vector myInterface = (Vector)myNode.get(address.toString());
        //System.out.println("getRemainingThroughput: for interface "
          //  + address + ", SL = " + service_level + " of node #"+
myNodeId);
        // if this interface has more service levels than this service
level
        if (myInterface.size() >= service_level) {
          slp_qos = (SLP_QoS)myInterface.elementAt(service_level);
          // update it with these new values
        //System.out.println("PIB: resourceIsAvailable: SLP " +
service_level
        //      + " is available for
delay>="+delay+",loss_rate>="+loss_rate
        //      + ",throughput<="+(slp_qos.targetThroughput-
slp_qos.observedUtilization));
          if (slp_qos.serviceLevel == service_level) {
              //slp_qos.observedDelay <= delay &&
              //slp_qos.observedLossRate <= loss_rate &&
            remainingThroughput = slp_qos.targetThroughput
                                - slp_qos.observedUtilization;
            //System.out.println("getRemainingThroughput:
targetThroughput = "
              //  +slp_qos.targetThroughput+", observedUtilization = "
              //        +slp_qos.observedUtilization);
          }
        }
      }
    }
    //System.out.println("getRemainingThroughput: remainingThroughput =
"
    //                   +remainingThroughput);
    return remainingThroughput;
  }


  /**Henry
   * Gets the unallocated throughput of the interface specified
   * @param   address  The interface address
   * @return  The amount throughput unallocated
   */
  public int getUnallocatedThroughput(IPv6Address address){
    //, int delay, int loss_rate, int throughput){
    int available_throughput =
Server.INITIALTHROUGHPUT;//Unallocated_allotment;
    Integer myNodeId = new Integer(0);
```

```
    Enumeration e = nodes.keys();
    // for each of the nodes in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if this node contains an interface with the address passed in
      if (myNode.containsKey(address.toString())){
        Vector myInterface = (Vector)myNode.get(address.toString());
        for (int service_level=Server.CONTROL_SERVICELEVEL;
          service_level<Server.NUMBEROFSERVICELEVELS; service_level++)
{
          // if this interface has more service levels than this
service level
          slp_qos = (SLP_QoS)myInterface.elementAt(service_level);
          //System.out.println("slp_qos = "+slp_qos);
          // update it with these new values
          available_throughput = available_throughput
                  - slp_qos.targetThroughput;
          //System.out.println("available throughput =
"+available_throughput);
        }//end for
      }
    }//end while
    //System.out.println("getUnallocatedThroughput:
"+available_throughput);
    return available_throughput;
  }


//**********************************************************************
********
// These methods are used to process a flow request from a host
//**********************************************************************
*******/

  /**
   * Finds a router id that has an interface to on the same link as the
host
   * making a flow request.
   * @param address The IPv6 address of the interface of the host
requesting
   * the flow.
   * @returns ARouter The router id of the first router found on this
link.
   */
  public int findARouterOnLink(IPv6Address address){
    int ARouter = 0;
    Integer myNodeId = null;
    // check to see if requesting host is a router itself
   Enumeration e = nodes.keys();
    // for each of the node ids in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      // if any of its address equals the address that was passed in
      if(myNode.containsKey(address.toString())){
        ARouter = myNodeId.intValue();
```

```
      }
    }
    // otherwise, find any other router on the same subnet
    if (ARouter == 0){
      e = nodes.keys();
      // for each of the node ids in the PIB
      while(e.hasMoreElements()){
        myNodeId = (Integer)e.nextElement();
        Hashtable myNode = (Hashtable)nodes.get(myNodeId);
        Enumeration addresses = myNode.keys();
        // for each of these interfaces
        while (addresses.hasMoreElements()){
          String nextAddress = (String)addresses.nextElement();
          try{
            // if this interface's network address equals that of the
link
            if
(IPv6Address.getByName(nextAddress).getNetworkAddress().toString().equa
ls(

address.getNetworkAddress().toString())){
              ARouter = myNodeId.intValue();
            }
          }catch(UnknownHostException uhe){
            gui.sendText(""+uhe);
          }
        }
      }
    }
    if (showComments){
      gui.sendText("PIB: findARouterOnLink: Router "
        + ARouter + " is found on same link " +
address.getNetworkAddress()
        + " as host " + address);
    }
    return ARouter;
  }

  /**
   * Determines if there is a path that can support a particular flow
request.
   * A value of zero is returned if no path can support this QoS.
   * @param source_router The node id of a router on the same physical
link as
   * the source host.
   * @param destination_router The node id of a router on the same
physical
   * link as the destination host.
   * @param myFlowRequest A host's request for the establishment of a
flow.
   * @returns path_id The id of a path that can support this request.
   */
  public int getPathThatCanSupportFlowRequest(int source_router,
                        int destination_router, FlowRequest
myFlowRequest){
    int path_id = 0;
    // for each path in the PIB
```

```
    Enumeration e_path_ids = paths.keys();
    while (e_path_ids.hasMoreElements()){
      Integer nextPathId = (Integer)e_path_ids.nextElement();
      Path nextPath = (Path)paths.get(nextPathId);
      // if it has the same source and destination router
      //   and its effective delay and loss rate is less than this
request
      //   and its throughput remaining is more than the requested
throughput
      if (nextPath.sourceRouter == source_router &&
          nextPath.destinationRouter == destination_router &&
          nextPath.effectiveDelay <= myFlowRequest.getRequestedDelay()
&&
          nextPath.effectiveLossRate <=
myFlowRequest.getRequestedLossRate() &&
          nextPath.effectiveThroughputRemaining
                                      >=
myFlowRequest.getRequestedThroughput()){
        path_id = nextPathId.intValue();
      }
    }
    if (showComments){
      if (path_id != 0){
        gui.sendText("PIB: getPathThatCanSupportFlowRequest: "
          + "Flow request from "+source_router+" to
"+destination_router
          +" with delay<="+myFlowRequest.getRequestedDelay()+", LR<="
          +myFlowRequest.getRequestedLossRate()+", RT>="
          +myFlowRequest.getRequestedThroughput()+"can be supported on
path: "
          + path_id);
      }
      else{
        gui.sendText("PIB: getPathThatCanSupportFlowRequest: "
          + "Flow request cannot be supported.");
      }
    }
    return path_id;
  }


  /**Henry
   * Determines if there is a path that can support a particular flow
request.
   * A value of -1 is returned if the destination is unreacheable.
   * A value of zero is returned if no path can support this QoS.
   * @param source_router The node id of a router on the same physical
link as
   * the source host.
   * @param destination_router The node id of a router on the same
physical
   * link as the destination host.
   * @param myFlowRequest A host's request for the establishment of a
flow.
   * @returns path_id The id of a path that can support this request.
   */
  public int getPathThatSupportFlowRequest(int source_router,
```

```
            int destination_router, FlowRequest myFlowRequest){
    System.out.println("\ngetPathsSupportingFlowRequest for: ");
    System.out.println("request: "+myFlowRequest);
    Vector supportingPaths = new Vector();
    Vector borrowablePaths = new Vector();
    int path_id = Server.FLOWNUNREACHEABLE;
    int bestPathId = path_id;
    Path bestPath = null;

    Hashtable possiblePaths = getAllPossiblePaths(source_router,
                                        destination_router);
    if (!possiblePaths.isEmpty()) {
      int requested_throughput =
myFlowRequest.getRequestedThroughput();
      bestPathId = Server.FLOWUNSUPPORTABLE;
      // for each possible path
      Enumeration e_path_ids = possiblePaths.keys();
      while (e_path_ids.hasMoreElements()){
        Integer nextPathId = (Integer)e_path_ids.nextElement();
        Path nextPath = (Path)possiblePaths.get(nextPathId);
        // for each slp
        for (int index = 0; index < nextPath.SLPSequence.size();
index++){
          ServiceLevelPipe nextSLP =
            (ServiceLevelPipe)nextPath.SLPSequence.elementAt(index);
          // if it has the same source and destination router
          //   and its effective delay and loss rate is less than this
request
          if (nextSLP.serviceLevel == myFlowRequest.getServiceLevel()
&&
            //nextPath.sourceRouter == source_router &&
            //nextPath.destinationRouter == destination_router &&
            nextPath.effectiveDelay <=
myFlowRequest.getRequestedDelay() &&
            nextPath.effectiveLossRate <=
myFlowRequest.getRequestedLossRate()){
            //find optimum path
            bestPathId = determineBestPath(nextPath,
nextPathId.intValue(),
                        bestPath, bestPathId);
            if (bestPathId == nextPathId.intValue()) {
              bestPath = nextPath;  //update bestPathId
              System.out.println("bestPath: "+bestPathId);

System.out.println("bestPath.effectiveThroughputRemaining= "
                  +bestPath.effectiveThroughputRemaining);
            }
            // if its throughput remaining is able to admit the
requested throughput
            if ((int)(bestPath.effectiveThroughputRemaining
                *loadingfactor[myFlowRequest.getServiceLevel()])
                >= myFlowRequest.getRequestedThroughput()){
              path_id = bestPathId;
            }
            //if increasing capacity can support request
            else if (resourceIsAvailable(bestPath, bestPathId,
              getIncreasableThroughput(bestPathId,
```

```
                myFlowRequest.getServiceLevel(),
bestPath.effectiveThroughputRemaining+requested_throughput))){
                if (!supportingPaths.contains(new Integer(bestPathId))){
                  supportingPaths.add(new Integer(bestPathId));
                }
              }
              else {
                if (!borrowablePaths.contains(new Integer(bestPathId))){
                  borrowablePaths.add(new Integer(bestPathId));
                }
              }
            }
          }//end for
        }//end while
        //if increasing capacity can support request
        if (path_id == Server.FLOWUNSUPPORTABLE && bestPath != null){
          //if increasing capacity can support request
          if (!supportingPaths.isEmpty()){
            //increase resource allotement
            path_id =
((Integer)supportingPaths.firstElement()).intValue();
            updateSLPTarget(myFlowRequest.getSourceInterface(),
              myFlowRequest.getServiceLevel(),
                myFlowRequest.getRequestedDelay(),
                myFlowRequest.getRequestedLossRate(),
                myFlowRequest.getRequestedThroughput());
          }
          //if inter-service borrowing can support request
          else if (!borrowablePaths.isEmpty()){
            //inter-service borrowing
            path_id =
((Integer)borrowablePaths.firstElement()).intValue();
            updateSLPTarget(myFlowRequest.getSourceInterface(),
              myFlowRequest.getServiceLevel(),
                myFlowRequest.getRequestedDelay(),
                myFlowRequest.getRequestedLossRate(),
                myFlowRequest.getRequestedThroughput());
          }
        }//end if
      }//end if

      if (showComments){
        if (path_id > Server.FLOWUNSUPPORTABLE){
          gui.sendText("PIB: getPathThatSupportFlowRequest: "
            + "Flow request from "+source_router+" to
"+destination_router
            +" with delay<="+myFlowRequest.getRequestedDelay()+", LR<="
            +myFlowRequest.getRequestedLossRate()+", RT>="
            +myFlowRequest.getRequestedThroughput()+"can be supported on
path: "
            + path_id);
        }
        else{
          gui.sendText("PIB: getPathThatSupportFlowRequest: "
            + "Flow request cannot be supported.");
        }
```

```
    }
    System.out.println("SupportingPaths: "+supportingPaths);
    System.out.println("BorrowablePaths: "+borrowablePaths);
    return path_id;
  }//end getPathThatSupportFlowRequest


  /**
   * Returns the incrementable throughput
   * @param   path_id The id of the path in question.
   * @param   service_level
   * @param   throughput
   * @return  The incrementable throughput
   */
  private int getIncreasableThroughput(int path_id,
              byte service_level, int throughput) {
    int allocated_throughput = getAllocatedThroughputOfAPath(path_id);
    float loaded_throughput =
loadingfactor[service_level]*allocated_throughput;
    double beta = getIncreasingFactor(throughput, loaded_throughput);
    System.out.println("allocated_throughput = "+loaded_throughput
      +", sum of throughput required = "+throughput+", beta = "+beta);
    double incremental_throughput = (1+beta)*loaded_throughput;
    System.out.println("getIncreasableThroughput: "+
incremental_throughput);
    return (int)incremental_throughput;
  }


  /**Henry
   * Return the factor to be used for increasing the resource
allocation
   * @param   throughput
   * @param   allocated_throughput
   * @return  The increasing factor to be used
   */
  private double getIncreasingFactor(int throughput, float
allocated_throughput) {
    if (allocated_throughput == 0) return 0;
    else return (throughput/allocated_throughput - 1d);
  }

  /**Henry
   * Returns the borrowable throughput
   * @param   path_id The id of the path in question.
   * @param   service_level
   * @param   throughput
   * @return  The borrowable throughput
   */
  private int getBorrowableThroughput(int path_id,
            byte service_level, int inter_service_level, int
throughput) {
    int allocated_throughput = getAllocatedThroughputOfAPath(path_id);
    float loaded_throughput =
loadingfactor[service_level]*allocated_throughput;
    double gamma = getBorrowingFactor(throughput, loaded_throughput);
    double borrowable_throughput = (1+gamma)*loaded_throughput;
```

```
    System.out.println("allocated_throughput = "+loaded_throughput
      +", sum of throughput required = "+throughput);
    return (int)(gamma*allocated_throughput);
  }

  /**Henry
   * Return the factor to be used for borrowing resources
   * @param   throughput
   * @param   allocated_throughput
   * @return  The borrowable factor to be used
   */
  private double getBorrowingFactor(int throughput, float
allocated_throughput) {
    if (allocated_throughput == 0) return 0;
    return 1d-((throughput+0.2*throughput)/allocated_throughput);
  }

  /**Henry
   * Checks if resources along a path is more than the throughput
specified
   * @param   path
   * @param   path_id
   * @param   throughput
   * @return  TRUE is unallocated throughput is more than that
specified
   */
  public boolean resourceIsAvailable(Path path, int path_id, int
throughput){
    int available_throughput = Server.INITIALTHROUGHPUT;
    int unallocated_throughput = 0;

    System.out.println("resourceIsAvailable: throughput =
"+throughput);
    IPv6Address[] addresses = getPathAddress(path);
    for (int i=0; i<addresses.length; i++) {
      unallocated_throughput = getUnallocatedThroughput(addresses[i]);
      if (available_throughput>unallocated_throughput) {
        available_throughput = unallocated_throughput;
        System.out.println("available throughput =
"+available_throughput);
      }
    }
    if (available_throughput > throughput){
      return true;
    }
    return false;
  }

  public boolean interServiceResourceIsAvailable(
        Path path, int path_id, int throughput){
    int available_throughput = Server.INITIALTHROUGHPUT;
    int unallocated_throughput = 0;

    System.out.println("resourceIsAvailable: throughput =
"+throughput);
    IPv6Address[] addresses = getPathAddress(path);
    for (int i=0; i<addresses.length; i++) {
```

```
      unallocated_throughput = getUnallocatedThroughput(addresses[i]);
      if (available_throughput>unallocated_throughput) {
        available_throughput = unallocated_throughput;
        System.out.println("available throughput =
"+available_throughput);
      }
    }
    if (available_throughput > throughput){
      return true;
    }
    return false;
  }

  /**Henry
   * Returns the path id of the best path between newPath and previous
bestPath
   * considering the type of algorithm used (WSPath or SWPath)
   * @param   newPath  The new path to be considered
   * @param   newPathId  The path id of the new path
   * @param   bestPath  The best path previously chosen
   * @param   bestPathId  The path id of the best path
   * @return  The path id of the result
   */
  private int determineBestPath(Path newPath, int newPathId,
                                Path bestPath, int bestPathId) {
    int pathSelection = WSPath;     //default
    if (bestPath == null) {   //not a valid path
      System.out.print("Throughput:
"+newPath.effectiveThroughputRemaining
            +", #ofHops: "+newPath.SLPSequence.size()+", the only");
      bestPath = newPath;
      bestPathId = newPathId;
    }
    else {
      if (pathSelection == WSPath) {      //Widest-Shortest Path
        if (newPath.effectiveThroughputRemaining >
            bestPath.effectiveThroughputRemaining) {
        System.out.print("Throughput:
"+newPath.effectiveThroughputRemaining
            +", #ofHops: "+newPath.SLPSequence.size()+", ");
        bestPath = newPath;
        bestPathId = newPathId;
        }
        else if (newPath.effectiveThroughputRemaining ==
          bestPath.effectiveThroughputRemaining &&
          newPath.SLPSequence.size() < bestPath.SLPSequence.size()) {
          System.out.print("Throughput:
"+newPath.effectiveThroughputRemaining
              +", #ofHops: "+newPath.SLPSequence.size()+", new");
          bestPath = newPath;
          bestPathId = newPathId;
        }
      }
      else {      //use Shortest-Widest Path
        //may be selected if rejection rate exceeds certain threshold
        if (newPath.SLPSequence.size() < bestPath.SLPSequence.size()) {
```

```
            System.out.print("Throughput:
"+newPath.effectiveThroughputRemaining
                +"#ofHops: "+newPath.SLPSequence.size()+", ");
            bestPath = newPath;
            bestPathId = newPathId;
          }
          else if (newPath.SLPSequence.size() ==
bestPath.SLPSequence.size() &&
            newPath.effectiveThroughputRemaining >
            bestPath.effectiveThroughputRemaining) {
            System.out.print("Throughput:
"+newPath.effectiveThroughputRemaining
                +", #ofHops: "+newPath.SLPSequence.size()+", ");
            bestPath = newPath;
            bestPathId = newPathId;
          }
        }
      }
      return bestPathId;
    }

    /*Henry
     * Returns all the possible paths between source_router and
     * destination_router in a Hashtable.
     * @param   source_router  The source node id
     * @param   destination_router  The destination node id
     * @return  Hashtable of all the possible paths
     */
    public Hashtable getAllPossiblePaths(int source_router,
                                         int destination_router){
      Hashtable pathArray = new Hashtable();
      // for each path in the PIB
      Enumeration e_path_ids = paths.keys();
      //System.out.println("getAllPossiblePaths: "+paths);
      while (e_path_ids.hasMoreElements()){
        Integer nextPathId = (Integer)e_path_ids.nextElement();
        Path nextPath = (Path)paths.get(nextPathId);
        if (nextPath.sourceRouter == source_router &&
            nextPath.destinationRouter == destination_router){
                pathArray.put(nextPathId, nextPath);
        }
      }
      //System.out.println("getAllPossiblePaths: pathArray="+pathArray);
      return pathArray;
    }


    /**Henry
     * Remove the flow_id given in the parameter from the PIB
     * @param   flow_id  The flow_id of the assigned flow
     */
    public void deleteAssignedFlow(int flow_id){
        Integer myflowId = new Integer(flow_id);
        Vector allPaths = getAllPathIds();
        for (int i=1; i<allPaths.size(); i++) {   //path ids start from 1
            // now assign this flow to this path
        Integer myPathId = (Integer)allPaths.elementAt(i);
```

```
          //System.out.println("PathId "+myPathId.toString());
          //Path myPath = (Path)allPaths.get(myPathId.intValue());
      Path myPath = (Path)paths.get(myPathId);
          //System.out.println("deleteAssignedFlow is checking path
"+myPath);
      if (myPath.flows.remove(myflowId) != null) {
            System.out.println("FlowId "+myflowId+ " removed");
        if (showComments){
          gui.sendText("PIB: deleteAssignedFlow: Flow "
            + flow_id + " is deleted from path.");
        }
            return;     //found
      }
    }
  }//end deleteAssignedFlow

  /**Xie
   * Finds an unassigned flow id and assigns this new flow to a path.
   * @param path_id The id of a path that can support this request.
   * @param source_router The node id of a router on the same physical
link as
   * the source host.
   * @param destination_router The node id of a router on the same
physical
   * link as the destination host.
   * @param myFlowRequest A host's request for the establishment of a
flow.
   * @returns max_flow_id The id that is being assigned to this flow.
   */
  public int getNewFlowId(int path_id, int source_router,
                          int destination_router, FlowRequest
myFlowRequest){
  /*
    int max_flow_id = 0;
    Enumeration e_path_ids = paths.keys();
    // for each path in the PIB
    while (e_path_ids.hasMoreElements()){
      Integer nextPathId = (Integer)e_path_ids.nextElement();
      Path nextPath = (Path)paths.get(nextPathId);
     Enumeration e_flow_ids = nextPath.flows.keys();
      // for each of the flows on this path
      while (e_flow_ids.hasMoreElements()){
        Integer nextFlowId = (Integer)e_flow_ids.nextElement();
        // if this flow id is greater than the current max flow id
        if (nextFlowId.intValue() > max_flow_id){
          max_flow_id = nextFlowId.intValue();
        }
      }
    }
    // now increment to get an unassigned flow id
    max_flow_id++;
  */
      // wrap around if necessary
      if (newAppFlowID >= MAX_FLOW_ID)
            newAppFlowID = MIN_APP_FLOW_ID;
      else
            newAppFlowID++;
```

```
    // now assign this flow to this path
    Path myPath = (Path)paths.get(new Integer(path_id));
    Flow_QoS myFlowQoS = new Flow_QoS();
    myFlowQoS.negotiatedDelay = myFlowRequest.getRequestedDelay();
    myFlowQoS.negotiatedDelay = myFlowRequest.getRequestedLossRate();
    myFlowQoS.negotiatedDelay = myFlowRequest.getRequestedThroughput();
    myPath.flows.put(new Integer(newAppFlowID),myFlowQoS);
    if (showComments){
      gui.sendText("PIB: assignNewFlowId: Flow "
        + newAppFlowID + " is assigned to path "+path_id);
    }
    return newAppFlowID;
  }

  /**Henry
   * Finds an unassigned flow id and assigns this new flow to a path.
   * @param path_id The id of a path that can support this request.
   * @param source_router The node id of a router on the same physical
link as
   * the source host.
   * @param destination_router The node id of a router on the same
physical
   * link as the destination host.
   * @param myFlowRequest A host's request for the establishment of a
flow.
   * @returns max_flow_id The id that is being assigned to this flow.
   */
  public int getANewFlowId(int path_id, int source_router,
                           int destination_router, FlowRequest
myFlowRequest){
    // wrap around if necessary
    if (newAppFlowID >= MAX_FLOW_ID)
      newAppFlowID = MIN_APP_FLOW_ID;
    else
      newAppFlowID++;

    // now assign this flow to this path
    Path myPath = (Path)paths.get(new Integer(path_id));
    Flow_QoS myFlowQoS = new Flow_QoS();
    myFlowQoS.negotiatedDelay = myFlowRequest.getRequestedDelay();
    myFlowQoS.negotiatedDelay = myFlowRequest.getRequestedLossRate();
    myFlowQoS.negotiatedDelay = myFlowRequest.getRequestedThroughput();
    myPath.flows.put(new Integer(newAppFlowID),myFlowQoS);
    if (showComments){
      gui.sendText("PIB: assignNewFlowId: Flow "
        + newAppFlowID + " is assigned to path "+path_id);
    }
    return newAppFlowID;
  }

  /**
   * Retrieves the sequence of SLP that make up a given path.
   * @param path_id The id of the path in question.
   * @returns slps_in_path A vector of SLPs that compose this path.
   */
  public Vector getSLPSequenceOfPath(int path_id){
```

```
    int sequenceNumber = 0;
    IPv6Address link_id = null;
    Vector slps_in_path = new Vector();
    // get the sequence of slps that compose the path
    Path path = (Path)paths.get(new Integer(path_id));
    // for each slp
    for (int index = 0; index < path.SLPSequence.size(); index++){
      ServiceLevelPipe nextSLP =
(ServiceLevelPipe)path.SLPSequence.elementAt(index);
      // instantiate a slp sequence object
      SLPSequence SLP_Sequence = new SLPSequence();
      // set the slp sequence values
      SLP_Sequence.setServiceLevel(nextSLP.serviceLevel);
      // find what node this slp is attached to...
      Enumeration e = nodes.keys();
      int node_id = 0;
      // for each of the node ids in the PIB
      while(e.hasMoreElements()){
        Integer myNodeId = (Integer)e.nextElement();
        Hashtable myNode = (Hashtable)nodes.get(myNodeId);
        if (myNode.containsKey(nextSLP.address.toString())){
          node_id = myNodeId.intValue();
          link_id = nextSLP.address.getNetworkAddress();
        }
      }

      SLP_Sequence.setSourceRouter(node_id);
      SLP_Sequence.setPathId(path_id);
      SLP_Sequence.setLinkId(link_id);
      SLP_Sequence.setSequenceNumber(sequenceNumber);
      if (showComments){
        gui.sendText("PIB: getSLPSequenceOfPath: adding
"+SLP_Sequence);
      }
      // add it to the vector
      slps_in_path.addElement(SLP_Sequence);
      // increment for the next slp
      sequenceNumber++;
    }
    if (showComments){
      gui.sendText("PIB: getSLPSequenceOfPath: SLP sequence of path
"+path_id
        +" has "+path.SLPSequence.size()+" hops.");
    }
    return slps_in_path;
  }

  /**Henry
   * Returns a vector of the SLPs that forms the path_id
   * @param   path_id  The path id
   * @return  The vector of the SLPs
   */
  public Vector getSLPAddressOfPath(int path_id){
    int sequenceNumber = 0;
    IPv6Address link_id = null;
    Vector slps_in_path = new Vector();
    // get the sequence of slps that compose the path
```

```
    Path path = (Path)paths.get(new Integer(path_id));
    // for each slp
    return slps_in_path = getSLPAddressOfPath(path);
  }

  /**Henry
   * Returns a vector of the SLPs that forms the path_id
   * @param   path  The path
   * @return  The vector of the SLPs
   */
  public Vector getSLPAddressOfPath(Path path){
    Vector slps_in_path = new Vector();
    // for each slp
    for (int index = 0; index < path.SLPSequence.size(); index++){
      ServiceLevelPipe nextSLP =
(ServiceLevelPipe)path.SLPSequence.elementAt(index);
      //SLP nextSLP = (SLP)path.SLPSequence.elementAt(index);
      // add it to the vector
      slps_in_path.addElement(nextSLP.address);
    }
    return slps_in_path;
  }

  /**Henry
   * Returns the IPv6Addresses of the SLP sequence that forms the
path_id
   * @param   path_id  The path id
   * @return  The IPv6Addresses of the SLP sequence
   */
  public IPv6Address[] getPathAddress(int path_id){
    //SLP nextSLP;
    IPv6Address[] pathAddress = null;
    Vector slps = getSLPAddressOfPath(path_id);
    if (!slps.isEmpty()) {
      pathAddress = new IPv6Address[slps.size()];
      for (int i=0; i<slps.size(); i++) {
        pathAddress[i] = (IPv6Address)slps.get(i);
        if (showComments){
        gui.sendText("PIB: getPathAddress: Interface " +
pathAddress[i]);
        }
      }
    }
    return pathAddress;
  }

  /**Henry
   * Returns the IPv6Addresses of the SLP sequence that forms the
path_id
   * @param   path_id  The path id
   * @return  The IPv6Addresses of the SLP sequence
   */
  public IPv6Address[] getPathAddress(Path path){
    //SLP nextSLP;
    IPv6Address[] pathAddress = null;
    Vector slps = getSLPAddressOfPath(path);
    if (!slps.isEmpty()) {
```

```
      pathAddress = new IPv6Address[slps.size()];
      for (int i=0; i<slps.size(); i++) {
        pathAddress[i] = (IPv6Address)slps.get(i);
        if (showComments){
          gui.sendText("PIB: getPathAddress: Interface " +
pathAddress[i]);
        }
      }
    }
    return pathAddress;
  }

  /**
   * Retrieves  the IPv6 address of an interface.
   * @param node_id The id of the router whose interface is being
queried.
   * @param link_id The network portion of the IPv6 address of an
interface.
   * @returns address The address of the interface that connects this
node and
   * link.
   */
  public IPv6Address getInterfaceAddress(int node_id, IPv6Address
link_id){
    IPv6Address address = new IPv6Address();
    IPv6Address tempAddress = null;
    String nextAddress;
    Hashtable node_interfaces = (Hashtable)nodes.get(new
Integer(node_id));
    Enumeration e_addresses = node_interfaces.keys();
    // for each of these interfaces
    while (e_addresses.hasMoreElements()){
      nextAddress = (String)e_addresses.nextElement();
      try{
        tempAddress = IPv6Address.getByName(nextAddress);
      }catch(UnknownHostException uhe){
        gui.sendText(""+uhe);
      }
      // if this interface's network address equals that of the link
      if
(tempAddress.getNetworkAddress().toString().equals(link_id.toString()))
{
        address = tempAddress;
      }
    }
    if (showComments){
      gui.sendText("PIB: getInterfaceAddress: Interface " + address
        + " connects " + node_id + " to link " + link_id);
    }
    return address;
  }

//*****************************************************************
********
// These methods are used to determine all possible paths across the
network
```

```
//*************************************************************************
*******/

  /**
   * Retrieve all of the router ids assigned by the PIB so far.
   * @returns V A vector of all assigned router ids.
   */
  public Vector getAllRouterIds(){
    Vector V = new Vector();
    Integer myNodeId;
    Enumeration e = nodes.keys();
    // for each node id in the PIB
    while(e.hasMoreElements()){
      myNodeId = (Integer)e.nextElement();
      // add it to the vector
      V.addElement(myNodeId);
    }
    if (showComments){
      gui.sendText("PIB: getAllRouterIds: All router ids returned:");
      gui.sendText(""+V);
    }
    return V;
  }


  /**
   * Retrieves the maximum service level of this SAAM region.
   * @returns max_slp_id The numerically highest service level id
assigned.
   */
  public int findMaxServiceLevel(){
    int max_slp_id = 0;
    Hashtable myNode = new Hashtable();
    Enumeration e1,e2;
    e1 = nodes.elements();
    // for each node in the PIB
    while(e1.hasMoreElements()){
      myNode = (Hashtable)e1.nextElement();
      e2 = myNode.elements();
      // for each interface of this node
      if (e2.hasMoreElements()){
        Vector myInterface = (Vector)e2.nextElement();
        // determine the maximum number of service levels
        if (max_slp_id < myInterface.size())
          max_slp_id = myInterface.size()-1;
      }
    }
    if (showComments){
      gui.sendText("PIB: findMaxServiceLevel: The max service level is
"
        + max_slp_id);
    }
    return max_slp_id;
  }

  /**
```

```
   * Retrieves an array of parents for each router.  A parent is a
directly
   * connected node.
   * @param V A vector of all router ids.
   * @param service_level The level of service for which paths are
being built
   * for.
   * @returns parent A hashtable of vectors containing the parents of
each router.
   */
  public Hashtable getParents(Vector V, int service_level){
    Hashtable node1 = new Hashtable();
    Hashtable node2 = new Hashtable();
    Vector myVector = new Vector();
    Enumeration e1_interface_ids,e2_nodes,
e2_node_ids,e3_interface_ids;
    IPv6Address link_id = null;
    Integer node_id;
    String address;
    Hashtable parent = new Hashtable();
    // for each "destination" node in vector V
    for (int index = 0; index < V.size(); index++){
      myVector = new Vector();
      // get this destination node's interfaces in order to know
      //   all of its directly connected links
      node1 = (Hashtable)nodes.get(V.elementAt(index));
      e1_interface_ids = node1.keys();
      // for each interface ( or directly connected link...)
      while (e1_interface_ids.hasMoreElements()){
        try{
          link_id = IPv6Address.getByName(

((String)e1_interface_ids.nextElement())).getNetworkAddress();
        }catch(UnknownHostException uhe){
          gui.sendText(""+uhe);
        }
        //get all nodes that that are also connected to the link
        e2_nodes = nodes.elements();
        e2_node_ids = nodes.keys();
        // for each node in the PIB
        while (e2_nodes.hasMoreElements()){
          node2 = (Hashtable)e2_nodes.nextElement();
          node_id = (Integer)e2_node_ids.nextElement();
          e3_interface_ids = node2.keys();
         // for each interface on this node
          while(e3_interface_ids.hasMoreElements()){
            address = (String)e3_interface_ids.nextElement();
           try{
             IPv6Address tempAddress =
IPv6Address.getByName(address).getNetworkAddress();
              // if this interface is also connected to this link
              //   and this node is not the "destination" node
              if ((link_id.toString().equals(tempAddress.toString()))
                                          &&
!node_id.equals(V.elementAt(index))){
                // add it to the parent vector of this "destination"
node
```

```
                myVector.addElement(node_id);
              } //end if
            }catch(UnknownHostException uhe){
              gui.sendText(""+uhe);
            }
          } // end while e3_interface_ids
        } // end while e2_nodes
      } // end while e1_interfaces
      parent.put(V.elementAt(index),myVector);
    } // end for
    if (showComments){
      gui.sendText("PIB: getParents: Parent hashtable returned:");
      gui.sendText(""+parent);
    }
    return parent;
  }

  /**
   * Assigns a path from a source and to a destination.
   * @param source_router The node_id of the source of the path.
   * @param destination_router The node_id of the destination of the
path.
   * @returns max_path_id The id to be assigned to this new path.
   */
  public int getNewPathId(int source_router,int destination_router){
    int max_path_id = 0;
    Integer path_id;
    Enumeration e_path_ids = paths.keys();
    // for each path in the PIB
    while (e_path_ids.hasMoreElements()){
      path_id = (Integer)e_path_ids.nextElement();
      // if this path id is greater than the max so far
      if (max_path_id < path_id.intValue())
        max_path_id = path_id.intValue();
    }
    // increment to get unassigned path id
    max_path_id++;
    Path path = new Path();
    path.sourceRouter = source_router;
    path.destinationRouter = destination_router;
    // add this new path to the PIB
    paths.put(new Integer(max_path_id),path);
    if (showComments){
      gui.sendText("PIB: getNewPathId: Path " + max_path_id + " from "
        +source_router+" to "+destination_router+" is inserted.");
    }
    return max_path_id;
  }

  /**
   * Identifies the id of the physical link between two adjacent
routers. If no
   * link exists between a source and destination router, the default
address
   * of all zeros is returned.
   * @param source_router The node_id of a router.
   * @param destination_router The node_id of an adjacent router.
```

```
   * @returns subnet The link id between this source and destination.
   */
  public IPv6Address getLinkBetween(int source_router, int
destination_router){
    IPv6Address subnet = new IPv6Address();
    String source_address, destination_address;
    Hashtable source_node = (Hashtable)nodes.get(new
Integer(source_router));
    Hashtable destination_node =
                        (Hashtable)nodes.get(new
Integer(destination_router));
    Enumeration e_source_interfaces = source_node.keys();
    // for each interface assigned to this source node in the PIB
    while (e_source_interfaces.hasMoreElements()){
      source_address = (String)e_source_interfaces.nextElement();
      Enumeration e_destination_interfaces = destination_node.keys();
      // for each interface assigned to this destination node in the
PIB
      while (e_destination_interfaces.hasMoreElements()){
        destination_address =

(String)e_destination_interfaces.nextElement();
        try{
          // if this destination interface equals this source interface
          if
(IPv6Address.getByName(source_address).getNetworkAddress().toString().e
quals(

IPv6Address.getByName(destination_address).getNetworkAddress().toString
()))){
            // get the network address
            subnet =
IPv6Address.getByName(source_address).getNetworkAddress();
          }
        }catch (UnknownHostException uhe){
          gui.sendText(""+uhe);
        }
      }
    }
    if (showComments){
      gui.sendText("PIB: getLinkBetween: Link between " + source_router
        + " and " + destination_router + " is " + subnet);
    }
    return subnet;
  }

  /**
   * Assigns a service level pipe sequence entry in the building of a
path.
   * @param service_level The level of service for which paths are
being built
   * for.
   * @param source_router The node_id of the source of the SLP.
   * @param link_id The subnet that this SLP goes over.
   * @param path_id The id assigned to the path.
   * @param sequence_number The number assigned to specify the sequence
of this
```

```
   * SLP in the path.
   */
  public void assignSLPSequence(int service_level, int source_router,
    IPv6Address link_id, int path_id, int sequence_number){
    // get this path object
    Path myPath = (Path)paths.get(new Integer(path_id));
    ServiceLevelPipe slp = new ServiceLevelPipe();
    slp.address = getInterfaceAddress(source_router, link_id);
    slp.serviceLevel = service_level;
    // add this slp to the sequence of slps in this path
    myPath.SLPSequence.insertElementAt(slp, sequence_number);
    if (showComments){
      gui.sendText("PIB: assignSLPSequence: SLP#"+service_level+" from
"
        + source_router + " to link " + link_id + " on path " + path_id
        + " is assigned sequence number " + sequence_number);
    }
  }

//*********************************************************************
********
// These methods are used to determine the effective QoS of paths
//*********************************************************************
*******/

  /**
   * Retrieves a vector of all path ids constructed by the PIB.
   * @returns path_ids All of the path ids known to the PIB.
   */
  public Vector getAllPathIds(){
    Vector path_ids = new Vector();
    Enumeration e_path_ids = paths.keys();
    // for each path
    while (e_path_ids.hasMoreElements()){
      // add it to vector
      path_ids.addElement(e_path_ids.nextElement());
    }
    if (showComments){
      gui.sendText("PIB: getAllPathIds: paths in PIB:");
      gui.sendText(""+path_ids);
    }
    return path_ids;
  }

  /**
   * Retrieves the SLPs that make up a given path.
   * @param path_id The id of the path in question.
   * @returns slps_in_path The SLPs that make up the path.
   */
  public Vector getSLPsOfPath(int path_id){
    Vector slps_in_path = new Vector();
    SLP mySLP = new SLP();
    int node_id = 0;
    Path path = (Path)paths.get(new Integer(path_id));
    // for each of the slps in this path
    for (int index = 0; index < path.SLPSequence.size(); index++){
```

```
      ServiceLevelPipe slp =
(ServiceLevelPipe)path.SLPSequence.elementAt(index);
      // set the values for delivery to the server
      mySLP.setServiceLevel(slp.serviceLevel);
      mySLP.setAddress(slp.address);
      Enumeration e = nodes.keys();
      // for each of the nodes in the PIB
      while(e.hasMoreElements()){
        Integer myNodeId = (Integer)e.nextElement();
        Hashtable myNode = (Hashtable)nodes.get(myNodeId);
        // if this node contains this interface IPv6 address
        if (myNode.containsKey(slp.address.toString()))
          node_id = myNodeId.intValue();
      }
      // with this node's id, get its interfaces
      Hashtable interfaces = (Hashtable)nodes.get(new
Integer(node_id));
      // get vector of slp_qos
      Vector slps = (Vector)interfaces.get(slp.address.toString());
      // get slp_qos for this particular slp
      SLP_QoS slp_qos = (SLP_QoS)slps.elementAt(slp.serviceLevel);
      // set QoS of this slp for delivery to the server
      mySLP.setAllocatedThroughput(slp_qos.targetThroughput);
      mySLP.setDelay(slp_qos.observedDelay);
      mySLP.setLossRate(slp_qos.observedLossRate);
      int observedThroughput = slp_qos.observedUtilization / 100
                                          *
slp_qos.targetThroughput /10;
      mySLP.setThroughput(observedThroughput);
      if (showComments){
        gui.sendText("PIB:getSLPsOfPath: path " + path_id + ": address
."
          +slp.address+", service_level "+slp.serviceLevel+" observed
values:");
        gui.sendText("PIB:getSLPsOfPath: D = "+slp_qos.observedDelay
          +"ms, LR = " +slp_qos.observedLossRate+"%, U = "
          + slp_qos.observedUtilization / 10
          + "%, T = "+observedThroughput+"kbps");
      }
      // add this slp to the vector for delivery to the server
      slps_in_path.addElement(mySLP);
    }
    if (showComments){
      gui.sendText("PIB: getSLPsOfPath: SLP in path " + path_id +":");
      gui.sendText(""+slps_in_path);
    }
    return slps_in_path;
  }

  /**Henry
   * Retrieves the SLPs that make up a given path.
   * @param path_id The id of the path in question.
   * @returns slps_in_path The SLPs that make up the path.
   */
  public Vector getSLPsOfAPath(int path_id){
    Vector slps_in_path = new Vector();
    SLP mySLP = new SLP();
```

```
    int node_id = 0;

    //System.out.println("inside getSlpsOfaPath getting the paths from
the paths table");
    Path path = (Path)paths.get(new Integer(path_id));

    //System.out.println("Paths are taken Now I will process each
path.");
    // for each of the slps in this path
    for (int index = 0; index < path.SLPSequence.size(); index++){
      ServiceLevelPipe slp =
(ServiceLevelPipe)path.SLPSequence.elementAt(index);
      // set the values for delivery to the server
      mySLP.setServiceLevel(slp.serviceLevel);
      mySLP.setAddress(slp.address);
      Enumeration e = nodes.keys();
      // for each of the nodes in the PIB
      while(e.hasMoreElements()){
        Integer myNodeId = (Integer)e.nextElement();
        Hashtable myNode = (Hashtable)nodes.get(myNodeId);
        // if this node contains this interface IPv6 address
        if (myNode.containsKey(slp.address.toString()))
          node_id = myNodeId.intValue();
      }
      //System.out.println("Node_id getSLPsOfAPath: " + node_id);
      // with this node's id, get its interfaces
      Hashtable interfaces = (Hashtable)nodes.get(new
Integer(node_id));
      //System.out.println("interfaces are taken from the nodes
table");
      //gui.sendText("interfaces are taken from the nodes table");
      // get vector of slp_qos
      Vector slps = (Vector)interfaces.get(slp.address.toString());
      //gui.sendText("SLPs getSLPsOfAPath: " + slps.size());
      // get slp_qos for this particular slp
      //gui.sendText("I am going to take SLPqos of "
+slp.serviceLevel);
      SLP_QoS slp_qos = (SLP_QoS)slps.elementAt(slp.serviceLevel);
      //gui.sendText("the qos of the SLP is "+ slp_qos.toString());
      //gui.sendText("SLPQos is taken from the slp.");
      // set QoS of this slp for delivery to the server
      mySLP.setAllocatedThroughput(slp_qos.targetThroughput);
      mySLP.setDelay(slp_qos.observedDelay);
      mySLP.setLossRate(slp_qos.observedLossRate);
      int observedThroughput = slp_qos.observedUtilization;
      mySLP.setThroughput(slp_qos.observedUtilization);
      if (showComments){
        gui.sendText("PIB:getSLPsOfAPath: path " + path_id + ": address
"
          +slp.address+", service_level "+slp.serviceLevel+" observed
values:");
        gui.sendText("PIB:getSLPsOfAPath: D = "+slp_qos.observedDelay
          +"ms, LR = " +slp_qos.observedLossRate+"%, U = "
          + slp_qos.observedUtilization / 10
          + "%, T = "+observedThroughput+"kbps");
      }
      // add this slp to the vector for delivery to the server
```

```
      slps_in_path.addElement(mySLP);
    }
    if (showComments){
      gui.sendText("PIB: getSLPsOfAPath: SLP in path " + path_id +":");
      gui.sendText(""+slps_in_path);
    }
    return slps_in_path;
  }

  /**Henry
   * Returns the amount of throughput allocated to a path.
   * @param path_id The id of the path in question.
   * @returns allocated_throughput The allocated throughput of a path.
   */
  public int getAllocatedThroughputOfAPath(int path_id){
    //  byte service_level, int delay, int loss_rate, int throughput){
    Vector slps_in_path = new Vector();
    SLP mySLP = new SLP();
    int allocated_throughput = 0;
    int minAllocatedThroughput = Server.INITIALTHROUGHPUT;

    slps_in_path = getSLPsOfAPath(path_id);
      //gui.sendText("SLPs: "+SLPs);

    for (int index2 = 0; index2 < slps_in_path.size(); index2++){
      //gui.sendText("Taking the slp "+index2);
      mySLP = (SLP)slps_in_path.elementAt(index2);
      //System.out.println("mySLP = "+mySLP);
      // find allocated_throughput throughput
      allocated_throughput = mySLP.getAllocatedThroughput();
      if (allocated_throughput < minAllocatedThroughput){
        minAllocatedThroughput = allocated_throughput;
        //System.out.println("minAllocatedThroughput =
"+minAllocatedThroughput);
      }
    }
    return minAllocatedThroughput;
  }

  /**Henry
   * Retrieves the SLPs that make up a given path.
   * @param path_id The id of the path in question.
   * @returns slps_in_path The SLPs that make up the path.
   */
  public void updateSLPsOfAPath(int path_id,
      byte service_level, int delay, int loss_rate, int throughput){
    Vector slps_in_path = new Vector();
    SLP mySLP = new SLP();
    int node_id = 0;

    //System.out.println("inside getSlpsOfaPath getting the paths from
the paths table");
    Path path = (Path)paths.get(new Integer(path_id));

    //System.out.println("Paths are taken Now I will process each
path.");
    // for each of the slps in this path
```

```
    for (int index = 0; index < path.SLPSequence.size(); index++){
      ServiceLevelPipe slp =
(ServiceLevelPipe)path.SLPSequence.elementAt(index);
      // set the values for delivery to the server
      slps_in_path = getSLPSequenceOfPath(path_id);
      for (int i=0; i<slps_in_path.size(); i++) {
        mySLP = (SLP)slps_in_path.get(i);
        if ((byte)mySLP.getServiceLevel()==slp.serviceLevel){
          mySLP.setDelay(delay);
          mySLP.setLossRate(loss_rate);
          mySLP.setAllocatedThroughput(throughput);
        }
        System.out.println("updated SLP = "+mySLP);
      }
    }
  }

  /**
   * Records the calculated effective quality of service parameters for
a
   * particular path.
   * @param path_id The id of the path in question.
   * @param effectiveDelay The effective delay that can be expected
when
   * transmiting a flow over this path.
   * @param effectiveLossRate The effective loss rate that can be
expected when
   * transmiting a flow over this path.
   * @param effectiveThroughputRemaining The effective throughput
capacity that
   * was not being used at last observation.
   */
  public void setEffectiveQoSOfPath(int path_id, int effectiveDelay,
    int effectiveLossRate, int effectiveThroughputRemaining){
    // get this path
    Path path = (Path)paths.get(new Integer(path_id));
    // set its effective QoS
    path.effectiveDelay = effectiveDelay;
    path.effectiveLossRate = effectiveLossRate;
    path.effectiveThroughputRemaining = effectiveThroughputRemaining;
    if (showComments){
      gui.sendText("PIB: setEffectiveQoSOfPath: path "+path_id
        +" effectively has D = " + effectiveDelay
        + "ms, LR = " + effectiveLossRate
        + "%, remaining T = " + effectiveThroughputRemaining+"kbps");
    }
  }


  //Henry
  /**
   * Records the calculated effective quality of service parameters for
a
   * particular path.
   * @param path_id The id of the path in question.
   * @param effectiveDelay The effective delay that can be expected
when
```

```
   * transmiting a flow over this path.
   * @param effectiveLossRate The effective loss rate that can be
expected when
   * transmiting a flow over this path.
   * @param effectiveThroughputRemaining The effective throughput
capacity that
   * was not being used at last observation.
   */
  public void updateEffectiveQoSOfPath(int path_id, int effectiveDelay,
    int effectiveLossRate, int effectiveThroughputRemaining){
    // get this path
    Path path = (Path)paths.get(new Integer(path_id));
    // set its effective QoS
    path.effectiveDelay = effectiveDelay;
    path.effectiveLossRate = effectiveLossRate;
    //path.effectiveThroughputRemaining = effectiveThroughputRemaining;
    path.effectiveThroughputRemaining =
path.effectiveThroughputRemaining
                    - effectiveThroughputRemaining;
    if (showComments){
      gui.sendText("PIB: updateEffectiveQoSOfPath: path "+path_id
        +" effectively has D = " + effectiveDelay
        + "ms, LR = " + effectiveLossRate
        + "%, remaining T = " + effectiveThroughputRemaining+"kbps");
    }
  }

  /**
   * Retrieves a vector of all path ids that travses the specified SLP.
   * @param address The IPv6 address of an interface.
   * @param service_level The level of service that this logical pipe
is
   * providing.
   * @returns path_ids All of the path ids that traverse this SLP.
   */
  public Vector getAllPathIdsThatTraverseSLP(IPv6Address address,
                                                          int
service_level){
    Vector path_ids = new Vector();
    Enumeration e_paths = paths.elements();
    Enumeration e_path_ids = paths.keys();
    // for each path
    while (e_paths.hasMoreElements()){
      Path path = (Path)e_path_ids.nextElement();
      Integer path_id = (Integer)e_path_ids.nextElement();
      // get the sequence of slps that make up this path
      Vector slps = path.SLPSequence;
      // for each of each of these slps
      for (int index = 0; index < slps.size(); index++){
        ServiceLevelPipe slp = (ServiceLevelPipe)slps.elementAt(index);
        // if this slp's interface address equals this address
        if (slp.address.equals(address)){
          // add it to the vector
          path_ids.addElement(path_id);
        }
      }
    }
```

```
    if (showComments){
      gui.sendText("PIB: getAllPathIdsThatTravseSLP: paths over
"+address
        +"'s service level #"+ service_level);
      gui.sendText(""+path_ids);
    }
    return path_ids;
  }

  /**Henry
   * Records the calculated effective quality of service parameters for
a
   * particular path.
   * @param path_id The id of the path in question.
   * @param effectiveDelay The effective delay that can be expected
when
   * transmiting a flow over this path.
   * @param effectiveLossRate The effective loss rate that can be
   * expected when transmiting a flow over this path.
   * @param effectiveThroughputRemaining The effective throughput
   * capacity that was not being used at last observation.
   */
  public void updateRemainingBWOfPath(int path_id, byte service_level,
                                      int requestedThroughput){
    IPv6Address[] pathAddress;
    int remainingThroughput = 0;
    // get this path
    Path path = (Path)paths.get(new Integer(path_id));
    pathAddress = getPathAddress(path_id);
    remainingThroughput = getRemainingThroughput(
                          pathAddress, service_level);
    System.out.println("RemainingThroughput: "+remainingThroughput);
    path.effectiveThroughputRemaining = remainingThroughput
                    - requestedThroughput;
    System.out.println("Updated RemainingThroughput: "
                                  +path.effectiveThroughputRemaining);
    if (showComments){
      gui.sendText("PIB: updateRemainingBWOfPath: path "+path_id
        + "%, remaining T = " +
path.effectiveThroughputRemaining+"kbps");
    }
  }

  /**Henry
   * Updates all paths that using the source interface
   * @param   source  The interface of the source
   * @param   service_level  The service level of the path
   * @param   requestedThroughput  The throughput allocated
   */
  public void updateRemainingBWOfAllPaths(IPv6Address source,
                byte service_level, int requestedThroughput){
    IPv6Address[] pathAddress;
    int remainingThroughput = 0;
    // get this path
    Vector allPath = getAllPathIdsThatTraverseSLP(source,
service_level);
    for (int i=0; i<allPath.size(); i++) {
```

```
      Integer path_id = (Integer)allPath.get(i);
      Path path = (Path)paths.get(path_id);
      pathAddress = getPathAddress(path_id.intValue());
      remainingThroughput = getRemainingThroughput(pathAddress,
service_level);
      //System.out.print(", RemainingThroughput:
"+remainingThroughput);
      path.effectiveThroughputRemaining = remainingThroughput
                                        - requestedThroughput;
      //System.out.println(", Updated RemainingThroughput:
"+path.effectiveThroughputRemaining);
      //System.out.print("PathId: "+path_id.toString());
      //System.out.println(" "+path.toString());
    }
  }


  /**Henry
   * Retrieves a vector of all path ids that travses the specified SLP.
   * @param address The IPv6 address of an interface.
   * @param service_level The level of service that this logical pipe
is
   * providing.
   * @returns path_ids All of the path ids that traverse this SLP.
   */
  public Vector getAllPathIdsThatTraverseSLP(IPv6Address address,
                                             byte /*int*/
service_level){
    Vector path_ids = new Vector();
    Enumeration e_paths = paths.elements();
    Enumeration e_path_ids = paths.keys();
    // for each path
    while (e_path_ids.hasMoreElements()){
      Path path = (Path)e_paths.nextElement();
      Integer path_id = (Integer)e_path_ids.nextElement();
    //System.out.println("PIB: getAllPathIdsThatTravseSLP: path_id = "
+path_id.toString());
      // get the sequence of slps that make up this path
      Vector slps = path.SLPSequence;
      // for each of each of these slps
      for (int index = 0; index < slps.size(); index++){
        ServiceLevelPipe slp = (ServiceLevelPipe)slps.elementAt(index);
        // if this slp's interface address equals this address
        if (slp.serviceLevel == service_level &&
          slp.address.equals(address)){
          // add it to the vector
          path_ids.addElement(path_id);
        }
      }
    }
    if (showComments){
      gui.sendText("PIB: getAllPathIdsThatTravseSLP: paths over
"+address
        +"'s service level #"+ service_level);
      gui.sendText(""+path_ids);
    }
```

```
    //System.out.println("PIB: getAllPathIdsThatTravseSLP:"
+path_ids.toString());
    return path_ids;
  }

//**********************************************************************
********
// These methods are used to retrieve various other info
//**********************************************************************
*******/

  /**
   * Retrieves a vector of all interface addresses attached to this
router.
   * @param node_id The node id of the router in question.
   * @returns IPv6Addresses The interface addresses of this node.
   */
  public Vector getRouterInterfaces(int node_id){
    Vector IPv6Addresses = new Vector();
    Hashtable myNode = (Hashtable)nodes.get(new Integer(node_id));
    Enumeration e = myNode.keys();
    // for each interface on this node
    while(e.hasMoreElements()){
      String myAddress = (String)e.nextElement();
      try{
        // add it to the vector
        IPv6Addresses.addElement(IPv6Address.getByName(myAddress));
      }catch(UnknownHostException uhe){
        gui.sendText(""+uhe);
      }
    }
    if (showComments){
      gui.sendText("PIB: getRouterInterfaces: interfaces of node "
        +node_id+" returned:");
      gui.sendText(""+IPv6Addresses);
    }
    return IPv6Addresses;
  }

  /**
   * Deletes a specified router from the PIB.
   * @param node_id The node_id of the router to be deleted.
   */
  public void deleteARouter(int node_id){
    nodes.remove(new Integer(node_id));
    if (showComments){
      gui.sendText("PIB: deleteARouter: node "+node_id+" deleted.");
    }
  }

  /**
   * this method is added by Hasan UYSAL
   * it deletes all the paths that use a certain interface
   */
  public void deletePathsTraversingInterface(Vector pathIds){
    for(int i=0;i<pathIds.size();i++){
      paths.remove(pathIds.elementAt(i));
```

```
    }
  }

  /**
   * Retrieves a vector of all physical link ids known to the PIB.
   * @returns v_links All of the known links in the network.
   */
  public Vector getAllLinkIds(){
    Vector v_links = new Vector();
    Enumeration e = links.keys();
    // for each interface on this node
    while(e.hasMoreElements()){
      String myAddress = (String)e.nextElement();
      try{
        // add it to the vector
        v_links.addElement(IPv6Address.getByName(myAddress));
      }catch(UnknownHostException uhe){
        gui.sendText(""+uhe);
      }
    }
    if (showComments){
      gui.sendText("PIB: getAllLinkIds: all link ids:");
      gui.sendText(""+v_links);
    }
    return v_links;
  }

  /**
   * Retrieves a vector of all routers attached to a specific phyical
link.
   * @param link_id The IPv6 address of the link in question.
   * @returns routerIds The ids of routers that are directly attached
to this
   * link.
   */
  public Vector findRoutersOnLink(IPv6Address link_id){
    Vector routerIds = new Vector();
    Enumeration e = nodes.keys();
    // for each of the node ids in the PIB
    while(e.hasMoreElements()){
      Integer myNodeId = (Integer)e.nextElement();
      Hashtable myNode = (Hashtable)nodes.get(myNodeId);
      Enumeration addresses = myNode.keys();
      // for each of these interfaces
      while (addresses.hasMoreElements()){
        String nextAddress = (String)addresses.nextElement();
        try{
          // if this interface's network address equals that of the
link
          if
(IPv6Address.getByName(nextAddress).getNetworkAddress().toString()

.equals(link_id.toString())){
            routerIds.addElement(myNodeId);
          }
        }catch(UnknownHostException uhe){
          gui.sendText(""+uhe);
```

```
        }
      }
    }
    if (showComments){
      gui.sendText("PIB: findRoutersOnLink:routers on
link"+link_id+":");
      gui.sendText(""+routerIds);
    }
    return routerIds;
  }

}//end of ClassObjectStructure
```

APPENDIX J – SAAM SERVER.SERVER.DIFFSERV PACKAGE CODE

```
//11Dec1999[Henry]     - Created

package saam.server.diffserv;

import saam.net.*;
import saam.util.*;

/**
 * The <em>SLS</em> class stores the ServiceLevelSpec parameters that
describes
 * that describe the type differential service class.
 */
public class SLS {

  /**
   *  SLS format:
   *    1   |   4    |   1   |                2/5                    |
   *  DSCP  Profile  Scope  Disposition of non-conforming traffic
   */

  /** The code point assigned to this particular SLS. */
  private byte DSCP = 0;

  /**
   *  DSCP field:
   *            Bit position
   *    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
   *   IN |        PHB        |   CU  |
   *
   *      IN  :  in (1) or out (0) of profile
   *     PHB  :  Per Hop Behavior
   *      CU  :  currently unused (reserved)
   *   Bit 1  :  Delay Priority
   *   Bit 2  :  Throughput Priority
   *   Bit 3  :  Loss Priority
   */

  public static final byte GOLD_CLASS = 112;  //01110000
  public static final byte SILVER_CLASS = 96;  //01100000
  public static final byte BRONZE_CLASS = 64;  //01000000

  private static final int GoldClass_profile = 1000;  //kbps
  private static final int GoldClass_lossRate = 1;  //packet
  private static final int GoldClass_delay = 1; //1ms
  private static final int SilverClass_profile = 500;  //kbps
  private static final int SilverClass_lossRate = 5;  //packet
  private static final int SilverClass_delay = 5;     //1ms
  private static final int BronzeClass_profile = 100;  //kbps
  private static final int BronzeClass_lossRate = 10;  //packet
  private static final int BronzeClass_delay = 10;    //1ms

  /** The loss rate associated to this particular SLS. */
  private int lossRate = BronzeClass_lossRate;
```

```
  /** The loss rate associated to this particular SLS. */
  private int delay = BronzeClass_delay;

  /** The maximum throughput for this SLS */
  private int profile = 0;

  /** The extend where this SLS is applicable */
  private byte scope = 0;

  /** The private class that contains the disposition info for this
SLS. */
  private Disposition disposition;

  /** The negotiated data rate planned for this SLS. */
  public static final byte DISCARD = 1;
  public static final byte REMARK = 2;
  public static final byte SHAPE = 3;

  /**
   * A private class.which store the action to be taken when the
profile
   * has been violated.
   */
  private class Disposition {
    byte DSCP = 0;
    int profile = 0;
    byte action = 0;

    public Disposition (byte type){
      action = type;
    }

    public Disposition (byte type, byte new_DSCP){
      action = type;
      DSCP = new_DSCP;
    }

    public Disposition (byte type, int target_profile){
      action = type;
      profile = target_profile;
    }

  }

  /**
   * Constructs a SLS object without any arguments.
   */
  public SLS(){
    this.DSCP = BRONZE_CLASS;
    this.disposition = new Disposition(DISCARD);
  }

  /**
   * Constructs a SLS object using the parameters that are passed.
   * @param   DSCP The DS code point for this SLS
   * @param   profile The max. throughput in Mbps allowed for this SLS
```

```
 * @param   scope The area which this SLS is applicable to.
 * @param   action The action to be taken when profile is exceeded
 */
public SLS(byte DSCP, int profile, byte scope, byte action){
  this.DSCP = DSCP;
  this.profile = profile;
  this.scope = scope;
  this.disposition = new Disposition(action);
}


/**
 * Construct a SLS given a DSCP that is presumed to be supported
 * or a default class profile will be assumed.
 * @param   DSCP  The DS code point of this SLS
 */
public SLS(byte DSCP){
  this.DSCP = DSCP;
  if (DSCP == GOLD_CLASS) {
    this.profile = GoldClass_profile;
    this.disposition = new Disposition(DISCARD);
  }
  else if (DSCP == SILVER_CLASS) {
    this.profile = SilverClass_profile;
    this.disposition = new Disposition(DISCARD);
  }
  else {
    this.profile = BronzeClass_profile;
    this.disposition = new Disposition(DISCARD);
  }
}


/**
 * Constructs a SLS object using the parameters that are passed.
 * @param   DSCP The DS code point for this SLS
 * @param   profile The maximum throughput negotiated
 * @param   lossRate  The maximum loss rate negotiated
 * @param   delay  The maximum delay negotiated
 */
public SLS(int profile, int lossRate, int delay){
  if (profile > SilverClass_profile &&
      lossRate < SilverClass_lossRate &&
        delay < SilverClass_delay) {
    this.DSCP = GOLD_CLASS;
    this.profile = GoldClass_profile;
    this.lossRate = GoldClass_lossRate;
    this.delay = GoldClass_delay;
    this.disposition = new Disposition(DISCARD);
  }
  else if (profile > BronzeClass_profile &&
      lossRate < BronzeClass_lossRate &&
        delay < BronzeClass_delay) {
    this.DSCP = SILVER_CLASS;
    this.profile = SilverClass_profile;
    this.lossRate = SilverClass_lossRate;
    this.delay = SilverClass_delay;
```

```
    this.disposition = new Disposition(DISCARD);
  }
  else {
    this.DSCP = BRONZE_CLASS;
    this.profile = BronzeClass_profile;
    this.lossRate = BronzeClass_lossRate;
    this.delay = BronzeClass_delay;
    this.disposition = new Disposition(DISCARD);
  }
}


/**
 * Constructs a SLS object using the parameters that are passed.
 * @param   DSCP The DS code point for this SLS
 * @param   profile The max. throughput in Mbps allowed for this SLS
 * @param   scope The area which this SLS is applicable to.
 * @param   action The action to be taken when profile is exceeded
 * @param   new_DSCP The new DS code point to use for action
 */
public SLS(byte DSCP, int profile, byte scope, byte action,
                        byte new_DSCP){
  this.DSCP = DSCP;
  this.profile = profile;
  this.scope = scope;
  this.disposition = new Disposition(action, new_DSCP);
}

/**
 * Constructs a SLS object using the parameters that are passed.
 * @param   DSCP The DS code point for this SLS
 * @param   profile The max. throughput in Mbps allowed for this SLS
 * @param   scope The area which this SLS is applicable to.
 * @param   action The action to be taken when profile is exceeded
 * @param   new_profile The throughput for reshaping action
 */
public SLS(byte DSCP, int profile, byte scope, byte action,
                        int new_profile){
  this.DSCP = DSCP;
  this.profile = profile;
  this.scope = scope;
  this.disposition = new Disposition(action, new_profile);
}


/**
 * Returns the DSCP of this Service Level Spec
 * @return  byte
 */
public byte getDSCP(){
  return DSCP;
}

/**
 * Returns the DispositionAction of this Service Level Spec
 * @return  byte
 */
```

```
public byte getDispositionAction(){
  return disposition.action;
}

/**
 * Returns the ActionByte of this Service Level Spec
 * @return  byte
 */
public byte getActionByte(){
  return disposition.DSCP;
}

/**
 * Returns the ActionInt of this Service Level Spec
 * @return  int
 */
public int getActionInt(){
  return disposition.profile;
}

/**
 * Returns the LossRate of this Service Level Spec
 * @return  int
 */
public int getLossRate(){
  return lossRate;
}

/**
 * Returns the Delay of this Service Level Spec
 * @return  int
 */
public int getDelay(){
  return delay;
}

/**
 * Returns the Profile of this Service Level Spec
 * @return  int
 */
public int getProfile(){
  return profile;
}

/**
 * Returns the Scope of this Service Level Spec
 * @return  byte
 */
public byte getScope(){
  return scope;
}

/**
 * Returns the byte array that represents this Service Level Spec
 * @return  byte[]
 */
public byte[] getSLSBytes(){
```

```
    byte[] bytes;
    bytes = Array.concat(this.DSCP,
      PrimitiveConversions.getBytes(this.profile));
    bytes = Array.concat(bytes, this.scope);
    bytes = Array.concat(bytes, this.disposition.action);
    if (disposition.action == REMARK) {
      bytes = Array.concat(bytes,getActionByte());
    }
    else if (disposition.action == SHAPE) {
      bytes = Array.concat(bytes,
          PrimitiveConversions.getBytes(getActionInt()));
    }
    return bytes;
  }

  /**
   * Generates the string representation of this service level spec.
   * @returns SLS The string representation of this service level spec.
   */
  public String toString(){

    String SLS = "\nSLS: DSCP = "+DSCP
      +", Profile = "+profile+"kbps, Scope = "+scope+
      ", Disposition.Action = "+disposition.action;
    if (disposition.DSCP != 0){
      SLS = SLS + ", Disposition.new_DSCP = "+disposition.DSCP;
    }
    else if (disposition.profile != 0){
      SLS = SLS + ", Disposition.new_Profile = "+disposition.profile;
    }
    return SLS;
  }

}//end of SLS class
```

```
//11Dec1999[Henry]    - Created
//
//
package saam.server.diffserv;
//

//
import saam.message.*;
//
import saam.util.*;
//
import java.util.*;
//

//
/**
//
 * The <em>SLSTable</em> class is used store a list of customers'
//
 * SLS and provide operations for maintaining the list.
 */
public class SLSTable extends Hashtable{

  /** The SLS to be added  */
  private   SLS sls;

  /** The amount of throughput that has been assigned. */
  private int throughput_utilized = 0;


  /**
   * Constructs a SLSTable object without any arguments.which
   * will create a new Hastable.
   */
  public SLSTable(){
  }


  /**
   * Adds a SLS that corresponds to the type of service class given
   * for the user specified in the parameters to this SLS Table
   * and update the amount of throughput utilized so far
   * @param   user
   * @param   service_class
   */
  public void addSLS(int user, String service_class){
    if (service_class.equals("Gold")) {
      this.sls = new SLS(SLS.GOLD_CLASS);
    }
    else if(service_class.equals("Silver")) {
      this.sls = new SLS(SLS.SILVER_CLASS);
    }
    else {
      this.sls = new SLS(SLS.BRONZE_CLASS);
    }
    put(new Integer(user), sls);  //store SLS
```

```
    throughput_utilized = throughput_utilized + sls.getProfile();
  }

  /**
   * Adds a SLS for the user specified in the parameters to this
   * SLS Table and update the amount of throughput utilized so far
   * @param   user  The int of the user
   * @param   service_class  The service class for the user
   * @param   profile  The throughput required for this SLS
   * @param   scope  The scope for this SLS
   * @param   action  The action to be taken when the profile is
   * exceeded.
   */
  public void addSLS(int user, byte service_class, int profile,
                     byte scope, byte action){
    this.sls = new SLS(service_class, profile, scope, action);
    put(new Integer(user), sls);  //store SLS
    throughput_utilized = throughput_utilized + profile;
  }

  /**
   * Adds a SLS for the user specified in the parameters to this
   * SLS Table and update the amount of throughput utilized so far
   * @param   user  The int of the user
   * @param   SLS  The new SLS to be added
   */
  public void addSLS(int user, SLS sls){
    this.sls = sls;
    put(new Integer(user), sls);  //store SLS
    throughput_utilized = throughput_utilized + sls.getProfile();
  }


  /**
   * Adds a SLS for the user specified in the parameters to this
   * SLS Table and update the amount of throughput utilized so far
   * @param   user  The int of the user
   * @param   service_class  The service class for the user
   * @param   profile  The throughput required for this SLS
   * @param   scope  The scope for this SLS
   * @param   action  The action to be taken when the profile is
   * exceeded.
   * @param   new_DSCP  The new DSCP to use for the action
   */
  public void addSLS(int user, byte service_class, int profile,
                     byte scope, byte action, byte new_DSCP){
    this.sls = new SLS(service_class, profile, scope, action,
                     new_DSCP);
    put(new Integer(user), sls);  //store SLS
    throughput_utilized = throughput_utilized + profile;
  }

  /**
   * Adds a SLS for the user specified in the parameters to this
   * SLS Table and update the amount of throughput utilized so far
   * @param   user  The int of the user
   * @param   service_class  The service class for the user
```

```
   * @param   profile  The throughput required for this SLS
   * @param   scope  The scope for this SLS
   * @param   action  The action to be taken when the profile is
   * exceeded.
   * @param   target_throughput  The throughput to be used to shape
   * the traffic
   */
  public void addSLS(int user, byte service_class, int profile,
              byte scope, byte action, int target_throughput){
    sls = new SLS(service_class, profile, scope, action,
                      target_throughput);
    put(new Integer(user), sls);  //store SLS
    throughput_utilized = throughput_utilized + profile;
  }

  /**
   * Get the SLS that matches the user passed in the parameter.
   * @param   user  The int of the user.
   * @return  The SLS of the user.
   */
  public SLS getSLS(int user){
    if (isEmpty()){
      System.out.println("SLS Table is empty.");
    }
    else{
      sls = (SLS)get(new Integer(user));
      if (sls == null){
        System.out.println("SLS for "+user+" is not in SLS Table.");
      }
    }
    return sls;
  }


  /**
   * Get the amount of throughput that have been assigned
   * @return  The amount of throughput that have been assigned
   */
  public int getAssignedThroughput(){
    return throughput_utilized;
  }

  /**
   * Retrieves the SLS contained in the SLS Table that matches the
   * user specified in the parameter, and update the
throughput_utilized.
   * @param   user  The int of the user.
   */
  public void deleteSLS(int user){
    if (isEmpty()){
      System.out.println("SLS Table is empty.");
    }
    else{
      sls = (SLS)get(new Integer(user));
      if (sls == null){
        System.out.println("SLS for "+user
```

```
                    +" is not in SLS Table.");
            "
        }
        else{
          //update throughput_utilized
          throughput_utilized = throughput_utilized - sls.getProfile();
          remove(new Integer(user));
        }
      }
    }


    /**
     * Generates the string representation of this SLS Table.
     * @returns SLS_table The string representation of this SLS Table.
     */
    public String toString(){
      return "\nSLSTable contains: "+super.toString();
    }

}//end SLS_Table
```

```
//11Dec1999[Henry]    - Created

package saam.server.diffserv;

import saam.message.*;
import saam.util.*;
import java.util.*;

/**
 * The <em>SLSDbase</em> class is used store a list of routers'
 * SLSTable and provide operations for maintaining the list.
 */
public class SLSDbase extends Hashtable{

  /** The SLSTable to be added  */
  private SLSTable  SLS_table;

  /** The GUI for displaying the SLSTables. */
  private SLSTableGui gui;

  /** The vector table used by the GUI. */
  private Vector tableGui;

  /** The vector of names stored in the vector table  */
  private Vector names = new Vector();

  /** The integer array that specifies the column widths for
  the table. */
  private int[] columnWidths = {60,120,60,60,60,60};

  /**
   * Constructs a SLSDbase object without any arguments.which
   * will create a new Hastable.
   */
  public SLSDbase() {
    names.add("NodeID");
    names.add("UserID");
    names.add("DSCP");
    names.add("Profile");
    names.add("Scope");
    names.add("Disposition");
    tableGui = new Vector();
  }


  /**
   * Adds a SLSTable for the router with the node_id specified
   * in the parameters and update the amount of throughput
   * utilized so far
   * @param   node_id  The id that identifies the router.
   * @param   sls_table  The SLSTable of the router.
   */
  public void addSLSTable(Integer node_id, SLSTable sls_table){
    //System.out.println("addSLSTable:");
    this.SLS_table = sls_table;
    if (get(node_id) == null) {
      //System.out.println("new SLSTableGui created");
```

```
    gui = new SLSTableGui("Node"+node_id.toString(),
          names, columnWidths);
    tableGui.add(node_id.intValue()-1, gui);
  }
  else {
    gui = (SLSTableGui)tableGui.elementAt(
            node_id.intValue()-1);
  }
  put(node_id, SLS_table);  //store SLS_Table
}

/**
 * Used to display the SLSTables
 */
public void displaySLSTable(){
  Integer node_id;
  if(!isEmpty()) {
    for(Enumeration e = keys(); e.hasMoreElements();){
      node_id = (Integer)e.nextElement();
      gui = (SLSTableGui)tableGui.elementAt(
             node_id.intValue()-1);
      gui.fillTable(getTable(node_id));
    }//end for
  }
}


/**
 * Get the SLSTable associated to the node_id specified in the
 * parameter.
 * @param   node_id  The node which the SLSTable belongs to
 * @return  A vector of the SLS
 */
public Vector getTable(Integer node_id){
  SLSTable entry;
  if(isEmpty()) return null;
  Vector table = new Vector(size());
  entry = getSLSTable(node_id);
  for(Enumeration e2 = entry.keys(); e2.hasMoreElements();){
    int skey = ((Integer)e2.nextElement()).intValue();
    SLS entry2 = (SLS)entry.getSLS(skey);
    Vector oneRow = new Vector();
    oneRow.add(node_id.toString());
    oneRow.add(""+skey);
    oneRow.add(""+entry2.getDSCP());
    oneRow.add(""+entry2.getProfile());
    oneRow.add(""+entry2.getScope());
    oneRow.add(""+entry2.getDispositionAction());
    table.add(oneRow);
  }//end for
  return table;
}//getTable()

/**
 * Get the SLSTable that matches the router's node_id passed
 * in the parameter.
 * @param   node_id  The id that identifies the router.
```

```
   * @return  The SLSTable that matches the router's node_id.
   */
  public SLSTable getSLSTable(Integer node_id){
    if (isEmpty()){
      //System.out.println("SLS Dbase is empty.");
      SLS_table = null;
    }
    else{
      SLS_table = (SLSTable)get(node_id);
      /*if (SLS_table == null){
        System.out.println("SLS Table for router node_id = "
          +node_id+" is not in SLS Dbase.");
      }*/
    }
    return SLS_table;
  }


  /**
   * Retrieves the SLSTable contained in the SLS Dbase that
   * matches the node_id specified in the parameter.
   * @param   node_id  The id that identifies the router.
   */
  public void deleteSLSTable(Integer node_id){
    if (isEmpty()){
      System.out.println("SLS Dbase is empty.");
    }
    else{
      SLS_table = (SLSTable)get(node_id);
      if (SLS_table == null){
        System.out.println("SLS Table for router node_id = "
          +node_id+" is not in SLS Dbase.");
      }
      else{
        remove(node_id);
      }
    }
  }


  /**
   * Generates the string representation of this SLS Dbase.
   * @returns The string representation of this SLS Table.
   */
  public String toString(){
    return "SLSDbase contains: " + super.toString();
  }

}//end SLSDbase class
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX K – SAAM CONTROL.CONTROLEXECUTIVE CLASS CODE

```
//23Feb2000[Henry]            - modified
// Feb 2000[akkoc]    - modified
// 01Aug99 [Vrable]   - Created

package saam.control;

import java.net.UnknownHostException;
import java.net.InetAddress;
import java.util.*;
import java.lang.*;

import saam.Translator;
import saam.router.*;
import saam.net.*;
import saam.message.*;
import saam.residentagent.*;
import saam.residentagent.router.*;

import saam.event.*;
import saam.util.SAAMRouterGui;
import saam.util.Array;

import saam.util.PrimitiveConversions;
import saam.EmulationTable;

import saam.server.*;

/**
 * The ControlExecutive maintains control over the event handling
mechanism
 * within the saam protocol stack by acting as a registrar for Objects
that
 * wish to communicate on Channels or emulated ports.<p>
 * The ControlExecutive receives all ResidentAgents destined for any of
the
 * router interfaces the ControlExecutive instantiates.  Before
instantiating these
 * agents, the ControlExecutive could be programmed to perform various
policy
 * adherence checks such as disallowing agents whose package name is
 * saam.control for example.  A check such as this would prevent agents
from
 * accessing the Channel class directly and circumventing the Channel
 * registration process.<p>
 * The ControlExecutive passes a copy of itself to each ResidentAgent
that it
 * instantiates, thus allowing the agent access to the
ControlExecutive's
 * public methods.  Through the use of these methods, ResidentAgents
can
 * register to talk on or listen to SAAM ports or Channels, request
flows,
```

```
 * send Messages or SAAMPackets, register as MessageProcessors, or
retrieve
 * various types of information from the ControlExecutive.<p>
 * The ControlExecutive also receives all Message Objects that are
destined
 * for this router.  A Hashtable of MessageProcessors is maintained to
 * determine which processor to pass an incoming Message to.<p>
 */
public class ControlExecutive
    implements MessageProcessor, SaamTalker, SaamListener{

  //some well-known UDP ports
  public static final int ECHO_PORT         = 7;
  public static final int DISCARD_PORT      = 9;
  public static final int DAYTIME_PORT      = 13;
  public static final int TIME_SERVER_PORT  = 37;
  public static final int DNS_PORT          = 53;
  public static final int WWW_HTTP_PORT     = 80;
  public static final int CHAT_PORT         = 531;

  //saam ports/channels
  public static final int HIGHEST_WELL_KNOWN_PORT       = 1023;
  public static final int MAX_PORT                      = 65531;
  public static final int SAAM_CONTROL_PORT
      = 8000;
  public static final int ROUTER_STATUS_CHANNEL                 =
80000;

  /**
   * The ControlExecutive registers with itself as a MessageProcessor
capable of
   * processing messages of the following types.
   */
  private static final String[] messageTypes =
    {"saam.message.InterfaceID",
     "saam.message.ServerID",
     "saam.message.FlowResponse",
     "saam.message.DemoHello",
     "saam.message.DCM",
     "saam.message.UCM",
     "saam.message.ParentNotification",
     "saam.message.TimeScale",
     "saam.message.ServiceLevelSpec",
     "saam.message.ResourceAllocation",
     "saam.message.InterfaceFailure",
     "saam.message.TestMessage",
     };

  private static final boolean ROUTER_UP = true;
  private static final boolean ROUTER_DOWN = false;


  /**
   * The initial status of the router is false.  As key router
components
   * are added, this status is updated to reflect the router's ability
   * to route packets.
```

```
     */
    private boolean routerStatus = ROUTER_DOWN;

    /*
     The following boolean variables represent the status of the elements
  necessary
     to stand up a router.
    */
    private boolean helloMessageReceived;
    private boolean arpCacheReady;
    private boolean flowRoutingTableReady;
    private boolean emulationTableReady;
    private boolean outboundInterfaceReady;

    private int interfaceCount;
    private int numberOfSchedulersPresent;
    private int nextInboundInterface;
    private SAAMRouterGui gui;
    private MainGui mainGui;
    private PacketFactory packetFactory;
    private PacketFactory packetFactoryOut;
    private TransportInterface transportInterface;
    private ResidentAgent arpCache;

    private static RoutingAlgorithm routingAlgorithm;

    private Interface currentInterface;

    private Object theLock= new Object();//critical section lock

    //added by Henry
    private Server server;
    private String sender;   //The sender of a flow agent

    //below are added by akkoc
    private AutoConfigurationExecutive autoConExec;
    private int timeScale;
    private static RouterBoundCtrlChTable rotBonConTable;
    private IPv6Address routerId= new IPv6Address();
    private static EmulationTable emTable;
    private boolean isServer = false;
    private ServerTable serverTable;


    //Hasan UYSAL
    LsaGenerator lsaGuy;

    /**
     * If a ServerID Message comes from the DemoStation,
     * the IPv6Address associated with that ServerID will
     * be set as the dest in the IPv6Header of packets sent out
     * on Server-bound flow, otherwise, the default IPv6Address
     * will be set as the dest.
     */
    // private IPv6Address serverIP = new IPv6Address();  //Akkoc removed

    /**
```

```
    * The ServerID will be sent by the DemoStation.  //Akkoc removed
    */
//  private ServerID serverID;   //Akkoc removed

  /**
   * The Vector that contains Interfaces that have been instantiated on
   * this router.  Default size = 4.
   */
  private Vector interfaces = new Vector(4);

  /**
   * The Vector of IDs for each interface that has been instantiated on
   * this router.  Default size = 4.
   */
  private Vector interfaceIDs = new Vector(4);

  /**
   * The Vector of talkers that have passed the registration process
and
   * are authorized to talk on channels.
   */
  private Vector activeTalkers = new Vector();

  /**
   * The Hashtable of ResidentAgents that have been instantiated by the
   * ControlExecutive.
   */
  private Hashtable agents = new Hashtable();

  /**
   * The Hashtable of ResidentAgentCustomers that have registered to
receive
   * ResidentAgent replacements as they arrive.
   */
  private Hashtable agentCustomers = new Hashtable();

  /**
   * The Hashtable of MessageProcessors that have registered with this
   * ControlExecutive.
   */
  private Hashtable messageProcessors =
    new Hashtable();

  /**
   * The Hashtable of Channels that have been instantiated by this
ControlExecutive.
   */
  private Hashtable activeChannels = new Hashtable();

  /**
   * The Hashtable of channels that a given SaamTalker is registered to
talk on.
   */
  private Hashtable channelsTalkerHas = new Hashtable();

  /**
```

```
   * The Hashtable of channels that a given SaamListener is registered
to listen on.
   */
  private Hashtable channelsListenerHas = new Hashtable();

  /**
   * The Hashtable of Objects that have requested flows.
   */
  private Hashtable flowRequestors = new Hashtable();

  /**
   * The Hashtable of Objects that have been assigned flows.
   */
  private Hashtable assignedFlows = new Hashtable();


  /**
   * Instantiates and sets up communication with all Objects that are
necessary
   * to allow the ControlExecutive to start receiving ResidentAgents
and Messages.
   */
  public ControlExecutive(){
    mainGui = new MainGui(this,"SAAM Router Prototype");
    gui = new SAAMRouterGui(toString());
    transportInterface =  new TransportInterface(this);
    //for receiving inbound packets
    packetFactory = new PacketFactory(this);
    packetFactoryOut = new PacketFactory();

    emTable =new EmulationTable();
    arpCache = new ARPCache();
    serverTable = new ServerTable(this);
    autoConExec = new AutoConfigurationExecutive(this);

    //this should eventually become a ResidentAgent
    routingAlgorithm = new RoutingAlgorithm(this, arpCache);

    //Here is where the ControlExecutive registers itself as a
MessageProcessor
    registerMessageProcessor(this);

    /* Enable Talking on the channel that the Translator is listening
on for
       router status updates.*/
    try{
      addTalkerToChannel(this, ROUTER_STATUS_CHANNEL);
    }catch(ChannelException ce){
      gui.sendText(ce.toString());
    }

    try{
      //Get ownership of the SAAM_CONTROL_PORT so other applications
      //cannot.  When the TransportInterface sees a packet destined
      //for this port, it will not forward the packet directly on
      //the SAAM_CONTROL_PORT, rather, it will forward the packet
```

```
      //to the PacketFactory on the
ProtocolStackEvent.PACKETFACTORY_CHANNEL
      monitorPort(this, SAAM_CONTROL_PORT);
      gui.setTextField("Monitoring emulated port "+SAAM_CONTROL_PORT);
    }catch(PortAccessDeniedException pade){
      gui.sendText(pade.toString());
    }

    //Hasan UYSAL
    lsaGuy = new LsaGenerator(this);

    mainGui.updateDisplay();
  }//ControlExecutive()

  /**
   * Returns the IPv6Address of the server controlling this router
   * @return The IPv6Address of the server controlling this router.
   */
/*  public IPv6Address getServerIP(){
    return serverIP;
  }   */ // akkoc removed

  /**
   * Returns the Servertable maintained by the router
   * @return ServerTable containing the entries
   */
  public synchronized ServerTable getServerTable(){
    return serverTable;
  }

  /**
   * Returns the Servertable maintained by the router
   * @return ServerTable containing the entries
   */
  public String getSender(){
    return sender;
  }

   /**
   * Returns the timesacle value for those requiring it
   * @return int timescale value
   */
  public int getTimeScale(){
    return timeScale;
  }
   /**
   * Returns the EmulationTable of the router
   * @return EmulationTable containing the entries
   */
  public synchronized EmulationTable getEmulationTable(){
    return emTable;
  }

  /**
   * To let know whether to behave as router or server for classes
   * requiring that information
   * @return boolean value
```

```
   */
  public boolean getIsServer(){
     return isServer;
  }

  /**
   * Returns the Router idd for the router
   * @return IPv6Address of the router .
   */
  public IPv6Address getRouterId(){
    return this.routerId;
  }

  /**
   * Hasan UYSAL
   * returns the reference to lsa generator
   */
  public LsaGenerator getGenerator(){
    return lsaGuy;
  }

  /**
   * Returns the status of the ARPCache.
   * @return The status of the ARPCache ResidentAgent.  (if the
ARPCache
   * contains an entry that corresponds to the next hop for Server-
bounded
   * flow, this method returns true).
   */
  public boolean getArpCacheStatus(){
    return arpCacheReady;
  }

  /**
   * Returns the status of the EmulationTable.
   * @return The status of the EmulationTable.  (if the EmulationTable
   * contains an entry that corresponds to the next hop for Server-
bound
   * flow, this method returns true).
   */
  public boolean getEmulationTableStatus(){
    return emulationTableReady;
  }

  /**
   * There are three tables in every SAAM router:  The ARPCache, the
EmulationTable,
   * and the FlowRoutingTable.  Each of these tables notifies the
ControlExecutive
   * when it is ready to serve the router.  When all of these tables
are ready and
   * a few other conditions are met, the ControlExecutive sends a
notification to
   * the Translator.
   * @param o The Object sending the update.
   * @param status The status of o.
   */
```

```
  public void updateCoreServiceStatus(
    Object o, boolean status){

    String className = o.getClass().getName();
    if(className.equals(
      "saam.residentagent.router.ARPCache")){
       arpCacheReady = status;
       updateRouterStatus();
    }else if (className.equals("saam.Translator")){
       emulationTableReady = status;
       updateRouterStatus();
    }
    //do nothing if another Object called this method
  }

 // methods below are modified/killed by [akkoc]
  /**
   * The FlowRoutingTable uses this method to notify the
ControlExecutive when it
   * is ready to serve the router.
   * @param o The Object sending the update.
   * @param serverBoundNextHop The IPv6Address of the next hop
associated with
   * Server-bound flow.
   */
 /* public void updateCoreServiceStatus(Object o, IPv6Address
serverBoundNextHop){

      String className = o.getClass().getName();
      this.serverBoundNextHop = serverBoundNextHop;
      boolean status = (serverBoundNextHop.equals(
          new IPv6Address()))? false:true;
      flowRoutingTableReady = status;
      //the following logic does not work for some reason...
      //I think it's the ArpCache.query method
      if(!arpCacheReady){
        if(routingAlgorithm.checkARPCache(
          new ARPCacheEntry(serverBoundNextHop))){
          arpCacheReady=true;
        }
      }
      updateRouterStatus();
  } */

  /**
   * Returns the IPv6Address representing the next hop associated with
   * Server-bound flow.
   * @return The IPv6Address representing the next hop associated with
   * Server-bound flow.
   */
 /* public IPv6Address getServerBoundNextHop(){
    return serverBoundNextHop;
  }   */

//Henry
  public void updateSLSTable(){
    mainGui.updateSLSTables();
```

```
  }

  public void updateRouterSLSTable(){
    mainGui.updateRouterSLSTable();
  }

//Henry
  public void updateFlowTable(Vector data){
    mainGui.updateFlowTables(data);
  }

  /**
   * Performs a series of checks to determine the status of the router
and
   * then updates the status accordingly.
   */
  private synchronized void updateRouterStatus(){

  // used only to define whether router is up or not
  // displayRouterStatus();
   /* String myAddress = null;
    try{
      myAddress = InetAddress.getLocalHost().getHostAddress();
    }catch(UnknownHostException uhe){}    */

    //compare local address with the address of the server.  If the
    //two addresses are the same and there is at least one interface
    //to process traffic, then the outbound interface for this server
is ready.

    //19Dec99[akkoc]  - logic belowhas been changed acc, to new
paradigm
    // if there is at leat one interfce ready then out bound interface
is ready
             //OLD
   /* if(myAddress.equals(serverID.getIPv4()) &&
      (interfaces.size()>=1)){
      outboundInterfaceReady=true;
    }else if
      //otherwise, compare the network portion of the IPv6Address of
the next
      //hop associated with Server-bound flow to the network portions
of the
      //IPv6Addresses of the interfaces on this router.  If there is a
match,
      //the outbound interface is ready.
    (routingAlgorithm.determineOutboundInterface(interfaces,
      serverBoundNextHop)!=null){
      outboundInterfaceReady=true;
    }    */
    //NEW

    if(interfaces.size()>=1){
      outboundInterfaceReady=true;
    }
    //if all the conditions are met, notify the Translator that the
router
```

```
    //is ready.
    if(helloMessageReceived && arpCacheReady && emulationTableReady &&
       outboundInterfaceReady){

      if(routerStatus==ROUTER_DOWN){
        //need to bring router status up since it satisfies all
criterias
            routerStatus=ROUTER_UP;
        //notifying Translator
        RouterStatusEvent  event = new
RouterStatusEvent(toString(),this,
                                    ROUTER_STATUS_CHANNEL,routerStatus);
        try{
          talk(event);
        }catch(ChannelException ce){
          gui.sendText(ce.toString());
        }
        lsaGuy.startAction();
        gui.sendText("Router is still UP NOW...");
      }else{
      routerStatus=ROUTER_DOWN;
      //bringRouterDown();
      gui.sendText("Router is still down...");
     }
   }
  }// end of method

  /**
   * Displays the current status of the router.
   */
  private void displayRouterStatus(){
    gui.sendText("\nCurrent Router Status:");
    gui.sendText(" helloMessageReceived:   "+helloMessageReceived);
    gui.sendText(" arpCacheReady:          "+arpCacheReady);
    gui.sendText(" flowRoutingTableReady:  "+
      flowRoutingTableReady);
    gui.sendText(" emulationTableReady:    "+emulationTableReady);
    gui.sendText(" outboundInterfaceReady: "+
      outboundInterfaceReady+
      "\n");
  }

  /**
   * In the SAAM architecture, traffic cannot be sent between hosts if
the
   * hosts are not assigned flows.  This method provides the mechanism
by
   * which hosts request flows from the SAAM server.  With the
information
   * provided in the parameters, this method constructs a
saam.message.FlowRequest.
   * It then sends that FlowRequest to the protocol stack for
transmission to
   * the SAAM server.  Note: If the requestor is not listening to a
port, the
   * requestor should first call the listenToRandomPort method for a
port assignment.
```

```
   * @param requestor The Object requesting the flow.
   * @param sourcePort The local port that requestor is listening on.
   * @param destHost The IPv6Address of the destination.
   * @param requestedDelay The amount of delay the requestor will
tolerate.
   * @param requestedLossRate The rate of loss the requestor will
tolerate.
   * @param requestedThroughput The amount of throughput the requestor
will tolerate.
   */
  public synchronized long requestFlow(ResidentAgent requestor,
        short sourcePort, IPv6Address destHost, int requestedDelay,
          int requestedLossRate, int requestedThroughput)
            throws FlowException{

      long timeStamp = System.currentTimeMillis();
      flowRequestors.put(new Long(timeStamp),requestor);
      IPv6Address sourceHost =
        ((InterfaceID)interfaceIDs.get(0)).getIPv6();
      FlowRequest request = new FlowRequest(
sourceHost,destHost,timeStamp,

requestedDelay,requestedLossRate,requestedThroughput);

      //FlowRequests travel on Server-bound flow.
      short destPort = (short)SAAM_CONTROL_PORT;
      gui.sendText("requestFlow to: source = "+sourceHost+", dest =
"+destHost);
      Vector channelsToServer = serverTable.getTable();
      for (int i=0; i<channelsToServer.size(); i++) {
            Vector channelToAServer = (Vector)channelsToServer.get(i);
            int serverBoundFlowId =
Integer.parseInt((String)channelToAServer.get(0));
            try{
            IPv6Address serverIPv6 =
IPv6Address.getByName((String)channelToAServer.get(2));
            gui.sendText("\t\tServerBoundFlowId =
"+serverBoundFlowId+", ServerIPv6 = "+serverIPv6);
              //send(this, request, MainServerBoundCtrlFlowID,
sourcePort,serverIP,destPort);
              send(this, request, serverBoundFlowId+1, sourcePort,
serverIPv6, destPort);
            }catch(FlowException fe){
              gui.sendText(fe.toString());
            }catch(UnknownHostException uhe){
              gui.sendText(uhe.toString());
            }
      }
      return timeStamp;
  }

  /**
   * In the SAAM architecture, traffic cannot be sent between hosts if
the
   * hosts are not assigned flows.  This method provides the mechanism
by
```

```
   * which hosts request flows from the SAAM server.  With the
information
   * provided in the parameters, this method constructs a
saam.message.FlowRequest.
   * It then sends that FlowRequest to the protocol stack for
transmission to
   * the SAAM server.  Note: If the requestor is not listening to a
port, the
   * requestor should first call the listenToRandomPort method for a
port assignment.
   * @param requestor The Object requesting the flow.
   * @param sourcePort The local port that requestor is listening on.
   * @param destHost The IPv6Address of the destination.
   * @param requestedDelay The amount of delay the requestor will
tolerate.
   * @param requestedLossRate The rate of loss the requestor will
tolerate.
   * @param requestedThroughput The amount of throughput the requestor
will tolerate.
   */
  public synchronized long requestFlow(ResidentAgent requestor,
   short sourcePort, IPv6Address destHost, int user_id, int
requestedDelay,
   int requestedLossRate, int requestedThroughput)
     throws FlowException{

      long timeStamp = System.currentTimeMillis();
      flowRequestors.put(new Long(timeStamp),requestor);
      IPv6Address sourceHost =
        ((InterfaceID)interfaceIDs.get(0)).getIPv6();
      //Request a DiffServ flow
      FlowRequest request = new FlowRequest( sourceHost, destHost,
timeStamp,
            user_id, requestedDelay, requestedLossRate,
requestedThroughput);

      //FlowRequests travel on Server-bound flow.
      short destPort = (short)SAAM_CONTROL_PORT;
      gui.sendText("requestFlow to: source = "+sourceHost+", dest =
"+destHost);
      Vector channelsToServer = serverTable.getTable();
      for (int i=0; i<channelsToServer.size(); i++) {
            Vector channelToAServer = (Vector)channelsToServer.get(i);
            int serverBoundFlowId =
Integer.parseInt((String)channelToAServer.get(0));
            try{
            IPv6Address serverIPv6 = IPv6Address.getByName(
                        (String)channelToAServer.get(2));
            gui.sendText("\t\tServerBoundFlowId = "+serverBoundFlowId+
                        ", ServerIPv6 = "+serverIPv6);
              //send(this, request, MainServerBoundCtrlFlowID,
sourcePort,serverIP,destPort);
              send(this, request, serverBoundFlowId+1,
                              sourcePort, serverIPv6, destPort);
            }catch(FlowException fe){
              gui.sendText(fe.toString());
            }catch(UnknownHostException uhe){
```

```
            gui.sendText(uhe.toString());
          }
      }
      return timeStamp;
  }

  /**
   * Objects that have been assigned flows can send Messages with this
method.
   * In order to use this method, Objects must first request a flow
using the
   * requestFlow method; and then receive a flow assignment from the
server
   * @param sender The Object sending the message.
   * @param message The subclass of saam.message.Message to be sent.
   * @param flowID The flow ID that has been assigned for traffic from
sender
   *        destined for destHost.
   * @param sourcePort The port on the local machine that sender is
listening on.
   * @param destHost The IPv6Address of the destination.
   * @param destPort The port to which destHost is listening.
   */
  public synchronized void send(Object sender, Message message,
    int flowID, short sourcePort, IPv6Address destHost,
    short destPort) throws FlowException{

      if(sender.getClass().getName().equals(
        "saam.residentagent.router.LsaGenerator")){
          sender = this;
      }
      gui.sendText("\nSending Message...");
      //PacketFactory packetFactory = new PacketFactory();

         packetFactoryOut.append(message);

         SAAMPacket saamPacket = null;
        try{
          saamPacket = new SAAMPacket(
          packetFactoryOut.getBytes());
        }catch(UnknownHostException uhe){
          throw new FlowException(sender+
           " Problem building packet "+flowID);
        }

      //call the send method that takes an IPv6Packet,
      //using the IPv6Packet constructed by the TransportInterface
      IPv6Packet v6Packet = transportInterface.buildIPv6Packet(sender,
        saamPacket,flowID,sourcePort,destHost,destPort);
      gui.sendText(">> Message payload size = " +
v6Packet.getPayload().length);
      try{
        saamPacket = new SAAMPacket(v6Packet.getPayload());
      }catch(UnknownHostException uhe){
        throw new FlowException(sender+
          " Problem building packet "+flowID);
      }
```

```
      send(sender,v6Packet);

  }


  /**
   *  this method is written by Hasan UYSAL
   *  it sends the extra LSA to the server
   */
  public void sendLSA(LinkStateAdvertisement lsa){

    packetFactory.append(lsa.getBytes());

    SAAMPacket saamPacket=null;
    try{
      saamPacket = new SAAMPacket(packetFactory.getBytes());
    }catch(UnknownHostException ex){
      gui.sendText("Can not create SAAMPacket ControlExecutive
sendLSA().");
      return;
    }

    Vector servers=serverTable.getServerEntries();
    Enumeration serversEnum=servers.elements();
    while(serversEnum.hasMoreElements()){
      ServerTableEntry
serverEntry=(ServerTableEntry)serversEnum.nextElement();
      int flowId=serverEntry.getFlowId();
      IPv6Address serverIp=serverEntry.getServerAddress();

      try{

send(this,saamPacket,flowId,(short)SAAM_CONTROL_PORT,serverIp,(short)SA
AM_CONTROL_PORT);
      }catch(FlowException fe){
        gui.sendText("Can not send SAAMPacket ControlExecutive
sendLSA()"+
          " to server with flowId "+flowId+" and IP
"+serverIp.toString());
      }
    }
  }//end sendLSA()


  /**
   * Objects that have been assigned flows can send DCM messages with
this method.
   * @param sender The Object sending the message.
   * @param message The DCM message to send
   * @param flowID The flow ID that has been assigned for traffic from
sender
   *        destined for destHost.
   * @param sourcePort The port on the local machine that sender is
listening on.
   * @param destHost The IPv6Address of the destination.
   * @param destPort The port to which destHost is listening.
   */
```

```
  public synchronized void sendDCM(Object sender, DCM message, int
flowID,
                          short sourcePort, IPv6Address destHost, short
destPort)
                                                              throws
FlowException{

    //PacketFactory packetFactory = new PacketFactory();
    packetFactoryOut.appendDCM(message);
    SAAMPacket saamPacket = null;
    long time = System.currentTimeMillis();
    byte numMessages= 1;
    SAAMHeader saamHeader = new SAAMHeader(time,numMessages);
    try{
      saamPacket = new SAAMPacket(saamHeader,
packetFactoryOut.getDCMBytes());
    }catch(Exception uhe){
        throw new FlowException(sender+ " Problem building packet
"+flowID);
    }
    //call the send method that takes an IPv6Packet,
    //using the IPv6Packet constructed by the TransportInterface
    IPv6Packet v6Packet = transportInterface.buildIPv6Packet(sender,

saamPacket,flowID,sourcePort,destHost,destPort);
    gui.sendText(">> DCM Message payload size = " +
v6Packet.getPayload().length);
    gui.sendText(">> DCM Message IN IPV6 from size is = " +
v6Packet.getBytes().length);

    send(sender,v6Packet);

  } //end of sendDCM

  /**
   * Objects that have been assigned flows can send UCM messages with
this method.
   * @param sender The Object sending the message.
   * @param message The UCM message to send
   * @param flowID The flow ID that has been assigned for traffic from
sender
   *        destined for destHost.
   * @param sourcePort The port on the local machine that sender is
listening on.
   * @param destHost The IPv6Address of the destination.
   * @param destPort The port to which destHost is listening.
   *
  public synchronized void sendUCM(Object sender, UCM message, int
flowID,
                          short sourcePort,IPv6Address destHost, short
destPort)
                                                              throws
FlowException{
    //PacketFactory packetFactory = new PacketFactory();
    packetFactory.appendUCM(message);
    SAAMPacket saamPacket = null;
    long time = System.currentTimeMillis();
```

```
    byte numMessages= 1;
    SAAMHeader saamHeader = new SAAMHeader(time,numMessages);
    try{
      saamPacket = new SAAMPacket(saamHeader,
packetFactory.getUCMBytes());
    }catch(Exception uhe){
        throw new FlowException(sender+
          " Problem building packet "+flowID);
    }
    //call the send method that takes an IPv6Packet,
    //using the IPv6Packet constructed by the TransportInterface
    IPv6Packet v6Packet = transportInterface.buildIPv6Packet(sender,

saamPacket,flowID,sourcePort,destHost,destPort);
    gui.sendText(">> UCM Message payload size = " +
v6Packet.getPayload().length);
    gui.sendText(">> UCM Message IN IPV6 from size is = " +
v6Packet.getBytes().length);
    send(sender,v6Packet);
  } //end of sendUCM
  */

  /**
   * Huseyin UYSAL
   *
   */
  public void sendUCM(byte noMes,byte [] bytes,int flowId,IPv6Address
dest){
    packetFactoryOut.appendUCM(noMes,bytes);
    SAAMPacket saamPacket=null;
    try{
      saamPacket=new SAAMPacket(packetFactoryOut.getBytes());
    }catch(UnknownHostException he){
      gui.sendText("Exception while creating saam packet at sendUCM.");
    }

    try{

send(this,saamPacket,flowId,(short)SAAM_CONTROL_PORT,dest,(short)SAAM_C
ONTROL_PORT);
    }catch(FlowException fe){
      gui.sendText("Colud not send the UCM to the Server "+(flowId-1));
    }
  }



  /**
   * Objects that have been assigned flows can send PN messages with
this method.
   * @param sender The Object sending the message.
   * @param message The PN message to send
   * @param flowID The flow ID that has been assigned for traffic from
sender
   *        destined for destHost.
   * @param sourcePort The port on the local machine that sender is
listening on.
```

```
   * @param destHost The IPv6Address of the destination.
   * @param destPort The port to which destHost is listening.
   */
  public synchronized void sendPN(Object sender, ParentNotification
message,
              int flowID, short sourcePort,IPv6Address destHost ,short
destPort)
                                                              throws
FlowException{

    //PacketFactory packetFactory = new PacketFactory();
    packetFactory.appendPN(message);
    SAAMPacket saamPacket = null;
    long time = System.currentTimeMillis();
    byte numMessages= 1;
    SAAMHeader saamHeader = new SAAMHeader(time,numMessages);
    try{
      saamPacket = new SAAMPacket(saamHeader,
packetFactory.getPNBytes());
    }catch(Exception uhe){
        throw new FlowException(sender+
          " Problem building packet "+flowID);
    }
    //call the send method that takes an IPv6Packet,
    //using the IPv6Packet constructed by the TransportInterface
    IPv6Packet v6Packet = transportInterface.buildIPv6Packet(sender,

saamPacket,flowID,sourcePort,destHost,destPort);
    gui.sendText(">> PN Message payload size = " +
v6Packet.getPayload().length);
    gui.sendText(">> PN Message IN IPV6 from size is = " +
v6Packet.getBytes().length);
    send(sender,v6Packet);
  }//end sendPN



  /**
   * Objects that have been assigned flows can send SAAMPackets with
this method.
   * In order to use this method, Objects must first request a flow
using the
   * requestFlow method; and then receive a flow assignment from the
server
   * @param sender The Object sending the message.
   * @param saamPacket The SAAMPacket to be sent.
   * @param flowID The flow ID that has been assigned for traffic from
sender
   *        destined for destHost.
   * @param sourcePort The port on the local machine that sender is
listening on.
   * @param destHost The IPv6Address of the destination.
   * @param destPort The port to which destHost is listening.
   */
  public synchronized void send(Object sender, SAAMPacket saamPacket,
    int flowID, short sourcePort, IPv6Address destHost,
    short destPort) throws FlowException{
```

```
      //call the send method that takes an IPv6Packet,
      //using the IPv6Packet constructed by the TransportInterface
      gui.sendText("Sending SAAM Packet...");
      IPv6Packet v6Packet = transportInterface.buildIPv6Packet(sender,
        saamPacket,flowID,sourcePort,destHost,destPort);
      send(sender,v6Packet);
  }//send()

  /**
   * Objects that have been assigned flows can send IPv6Packets with
this method.
   * In order to use this method, Objects must first request a flow
using the
   * requestFlow method; and then receive a flow assignment from the
server.
   * note: If the packet is destined for an Interface that is one this
router,
   * the packet will be delivered without flow id verification.
   * @param sender The Object sending the message.
   * @param ipv6Packet The IPv6Packet to be sent.
   */
  public synchronized void send(Object sender, IPv6Packet ipv6Packet)
    throws FlowException{

    int flowID = ipv6Packet.getHeader().getFlowLabel();
    //verify that the sender owns the flowID
    gui.sendText("Sending IPv6 Packet on flow "+flowID);
    if(!routingAlgorithm.isApplicationLayerPacket(ipv6Packet)){
      ResidentAgent agent = (ResidentAgent)assignedFlows.get(
            new Integer(flowID));
      if(sender.getClass().getName().equals("saam.server.Server")
        || (agent!=null&&agent.equals(sender)) || sender.equals(this)){
        //Here we forward the packet to an inbound interface
        //so it will be processed just as if it were an inbound
        //packet.
        ProtocolStackEvent event = new ProtocolStackEvent(
          sender.toString(),this,
          ProtocolStackEvent.getFromNICToInterfaceChannel(
            nextInboundInterface),
             ipv6Packet.getBytes());
        try{
          gui.sendText("\n>> Enqueuing packet for transmission at
channel" +
            event.getChannel_ID());
          gui.sendText("  >> nextInboundInterface = " +
nextInboundInterface+" flowid is "+flowID);
          talk(event);
        }catch(ChannelException ce){
          gui.sendText(ce.toString());
        }
//        routingAlgorithm.routeInboundPacket(ipv6Packet.getBytes());
        nextInboundInterface++;
        if(nextInboundInterface>=interfaces.size()){
          nextInboundInterface=0;
        }
      }else {
```

```
        gui.sendText(sender+
          " doesn't own flow "+flowID);
        throw new FlowException(sender+
          " doesn't own flow "+flowID);
      }
    }else{
      //this packet is destined for an Interface on this router, to go
outside
      //so we forward it to the TransportInterface for delivery
      //on the proper emulated UDP port.

      IPv6Header v6Header = ipv6Packet.getHeader();

if(v6Header.getSource().toString().equals(IPv6Address.DEFAULT_HOST)){

v6Header.setSource(((Interface)interfaces.get(0)).getID().getIPv6());
        ipv6Packet.setHeader(v6Header);
      }

      //gui.sendText("received app. layer packet from application
layer");
      gui.sendText("\nforwarding to TransportInterface");

      SAAMPacket saamPacket = null;
      try{
        saamPacket = new SAAMPacket(ipv6Packet.getPayload());


      }catch(UnknownHostException uhe){
        gui.sendText(uhe.toString());
      }
      ProtocolStackEvent event = new ProtocolStackEvent(
        sender.toString(),
        //here we trick the TransportInterface into thinking this
        //event came from the routingAlgorithm.  Also, since the
        //routingAlgorithm is already registered to talk on this
        //channel, we make it past the security check that ensures
        //the talker is registered on the channel.
        routingAlgorithm,ProtocolStackEvent.
          FROM_ROUTINGALGORITHM_TO_TRANSPORTINTERFACE_CHANNEL,
        ipv6Packet.getBytes());
      try{

        talk(event);

      }catch(ChannelException tde){
        gui.sendText(tde.toString());
      }

    }


  }//send()

  /**
   * In order to send a flowRequest or any other type of traffic in a
SAAM network,
```

```
   * sending Objects must be listening to a port.  This method assigns
Objects a
   * random port that is higher than the highest well-known port, but
no higher than
   * MAX_PORT.
   * @param listener The SaamListener requesting a random port.
   */
  public int listenToRandomPort(SaamListener listener){
    int port = listenToRandomChannel(
      listener, HIGHEST_WELL_KNOWN_PORT+1,MAX_PORT);
    try{
      //the TransportInterface must be able to talk on the new port in
order
      //to deliver traffic to the listener
      addTalkerToChannel(transportInterface, port);
    }catch(ChannelException ce){
      gui.sendText(ce.toString());
    }
    return port;
      //also register the TransportInterface as a talker on
      //this port.
  }//listenToRandomPort()

  /**
   * Ports reside on Channels 0-MAX_PORT; any Channel higher than
MAX_PORT
   * is a Channel that is not associated with a port.  Examples of this
are
   * the communications Channels within the protocol stack.
   * @param listener The SaamListener requesting a random channel.
   * Note: Since this method is private, only the ControlExecutive can
   * assign listeners to a random channel.
   * @param lowestChannel The lowest channel in the range to be
selected from.
   * @param highestChannel The highest channel in the range to be
selected from.
   */
  private int listenToRandomChannel(
    SaamListener listener, int lowestChannel,
      int highestChannel){
    int channelFound = 0;
    boolean exception = true;
    while(exception){
      try{
        channelFound = (lowestChannel+(
          new Random()).nextInt(highestChannel-lowestChannel));
        addListenerToChannel(listener, channelFound);
        exception = false;
      }catch(Exception e){}
    }//while()
    return channelFound;
  }//listenToRandomChannel()

  /**
   * Returns the array of Strings representing the class
   * names of the messages this Object will register to process.
   * @return The array of Strings representing the class names
```

```
   * of the messages this Object will register to process.
   */
  public synchronized String[] getMessageTypes(){
    return messageTypes;
  }//getMessageTypes()

  /**
   * This private method is used by the ControlExecutive to perform the
   * steps necessary to instantiate an Interface.
   * @param id The InterfaceID that will be assigned to the interface
being
   * instantiated.  If an Interface with this id is already up, the
request
   * will be ignored.
   */
  private void standUpInterface(InterfaceID id){
    boolean alreadyActive = false;
    for (int i=0;i<interfaceIDs.size();i++){
      if(((InterfaceID)interfaceIDs.get(i)).equals(id)){
        alreadyActive = true;
        break;
      }
    }

    if(!alreadyActive){
      interfaceIDs.add(id);
      gui.sendText(" Instantiating interface["+interfaceCount++ +"]");
      gui.sendText(" IPv6Address: "+id.getIPv6().toString());

      Interface thisInterface = new Interface(this, id);
      interfaces.add(thisInterface);
      updateRouterStatus();
      routingAlgorithm.addInterface(thisInterface);
      try{
        addTalkerToChannel(this,
          ProtocolStackEvent.getFromNICToInterfaceChannel(
            interfaces.size()-1));
      }catch(ChannelException ce){
        gui.sendText(ce.toString());
      }
    }else{
      gui.sendText("Interface already active...");
    }

    //[Akkoc] added for router_id determination
    // to check whether this new interface can be routerId
    // Highest IPv6Adress has been taken as routerId
    byte[] candidate = id.getIPv6().getAddress();
    byte[] rId = routerId.getAddress();

      for(int i=0; i<rId.length; i++){
        if(candidate[i] < rId[i]) {
           break;
        }else if(candidate[i] > rId[i]) {
           routerId = id.getIPv6();
        }//end else if
      } // end for
```

```
      gui.sendText("Now router id is "+ getRouterId().toString());


  }//standUpInterface()

  /**
   * This method contains the logic needed by the ControlExecutive
   * to process the Messages it is registered to process.
   * @param message The subclass of saam.message.Message to be
processed.
   */
   //synchronized
  public void processMessage(Message message){
    String name = message.getClass().getName();

    if(name.equals(messageTypes[0])){
      InterfaceID id = (InterfaceID)message;
      standUpInterface(id);
      updateRouterStatus();
    }else if(name.equals(messageTypes[1])){
      try{
      //  serverID = ((ServerID)message); // akkoc killed
      //  serverIP = serverID.getIPv6(); //  akkoc killed
      }catch(Exception e){
        gui.sendText("Error processing serverID: "+e.toString());
      }
    }else if(name.equals(messageTypes[2])){
      gui.sendText("Got a FlowResponse.. ");
      FlowResponse response = (FlowResponse)message;
      long timeStamp = response.getTimeStamp();
      gui.sendText("TimeStamp: "+timeStamp);
      int flowID = response.getFlowId();
      gui.sendText("flowID: "+flowID);
      ResidentAgent requestor =
        (ResidentAgent)flowRequestors.get(new Long(timeStamp));
      gui.sendText("Forwarding to Requestor: "+requestor);
      if(requestor!=null){
        assignedFlows.put(new Integer(flowID),requestor);

        requestor.receiveFlowResponse(response);
      }//else do nothing


    }else if(name.equals(messageTypes[3])){
      //received a saam.message.DemoHello from the DemoStation
      gui.sendText("received message type: " + messageTypes[3]);
      //
      Vector helloInterfaces = ((DemoHello)message).getInterfaceIDs();
      for(int i=0;i<helloInterfaces.size();i++){
        InterfaceID id = (InterfaceID)helloInterfaces.get(i);
        standUpInterface(id);
      }
      helloMessageReceived = true;  //DemoHello - CRC
      updateRouterStatus();
      mainGui.updateDisplay();
    }
```

```
    // below else cases are added by [akkoc]
    else if(name.equals(messageTypes[4])){
      gui.sendText("received message type: " + messageTypes[4]);
      DCM receivedDcm = (DCM)message;
      gui.sendText("Sending the message to autoConfig.");
      autoConExec.processDCM(receivedDcm);
      gui.sendText("DCM processed.");

    }//end of else if for DCM
    else if(name.equals(messageTypes[5])){
      gui.sendText("received message type: " + messageTypes[5]);
      UCM ucm =(UCM) message;
      autoConExec.processUCM(ucm);

    }//end of elseif for ucm
    else if(name.equals(messageTypes[6])){ //Parent Notification
      gui.sendText("\n received message type: " + messageTypes[6]);
      ParentNotification pn = (ParentNotification)message;
      autoConExec.processPN( pn );

    }//end of elseif for parent notification
    else if(name.equals(messageTypes[7])){ //Parent Notification
      gui.sendText("\n received message type: " + messageTypes[7]);
      TimeScale ts = (TimeScale)message;
      timeScale = ts.getTimeScale();

    }//end of else if for parent notification
    else if(name.equals("ServiceLevelSpec")){   //ServiceLevelSpec
message
      gui.sendText("Got a ServiceLevelSpec.. ");
      //Router do nothing yet
    }
    else if(name.equals("saam.message.TestMessage")){
      gui.sendText("Got a TestMessage.. ");
      TestMessage testMsg = (TestMessage)message;
      sender = testMsg.getSender();
    }
    else if(name.equals(messageTypes[10])){
      //this is an interface failure message
      gui.sendText("A InterfaceFailure message arrived..");
      //find the interface that is failing
      Enumeration enum = interfaces.elements();
      InterfaceFailure failure=(InterfaceFailure)message;
      while(enum.hasMoreElements()){
        Interface iFace = (Interface)enum.nextElement();
        InterfaceID id = iFace.getID();
        IPv6Address ip = id.getIPv6();

        if(ip.equals(failure.getIP())){
          System.out.println("Interface with ip "+ip.toString()+" is
DOWN.");
          iFace.setState(this);
          return;
        }
      }
```

```
      System.out.println("There is no interface with ip
"+failure.getIP().toString());
    }

  }//processMessage()


  public void processMessage(byte[] bytes, String message){
    gui.sendText("\nprocessing Message: "+ message);
      if (server == null) {
            try {
                if(message.equals("FlowResponse")){
                  FlowResponse response = new FlowResponse(bytes);
                  gui.sendText("Received Message: "+ response);
                  long timeStamp = response.getTimeStamp();
                  gui.sendText("TimeStamp: "+timeStamp);
                  int flowID = response.getFlowId();
                  gui.sendText("flowID: "+flowID);
                  ResidentAgent requestor =
                    (ResidentAgent)flowRequestors.get(new
Long(timeStamp));
                  gui.sendText("Forwarding to Requestor: "+requestor);
                  if(requestor!=null){
                    assignedFlows.put(new Integer(flowID),requestor);

                    requestor.receiveFlowResponse(response);
                  }//else do nothing
                  }
         else if(message.equals("ResourceAllocation")){
                  ResourceAllocation ra = new
ResourceAllocation(bytes);
                  gui.sendText("Got a ResourceAllocation.. "+ra);
                  //do nothing yet
               }
                  else {
                  gui.sendText("Unknown Router bound message.");
                  }
          }
            catch (UnknownHostException uhe) {
                  gui.sendText("Router processMessage had
UnknownHostException:"+uhe);
            }
      }
      else {
            try {
         if(message.equals("FlowRequest")){
             FlowRequest request = new FlowRequest(bytes);
           gui.sendText("Received Message: "+ request);
           gui.sendText(
               "Calling Server method: processFlowRequest()");
           server.processFlowRequest((FlowRequest)request);
         }
         else if(message.equals("FlowTermination")){
            FlowTermination myFlow = new FlowTermination(bytes);
            gui.sendText("Received Message: "+ myFlow);
            gui.sendText(
              "Calling Server method: receiveFlowTermination()");
```

```
            server.receiveFlowTermination(myFlow.getFlowId());
         }
         else if(message.equals("SLSTableEntry")){
            SLSTableEntry slsEntry = new SLSTableEntry(bytes);
            gui.sendText("Calling Server method:
receiveSLSTableUpdate()");
            server.receiveSLSTableUpdate(slsEntry);
         }
         else if(message.equals("ResourceAllocation")){
            ResourceAllocation ra = new ResourceAllocation(bytes);
            gui.sendText("Got a ResourceAllocation.. "+ra);
            //do nothing yet
         }
                  else {
                   gui.sendText("Unknown Server bound message.");
                  }
            }
            catch (UnknownHostException uhe) {
                 gui.sendText("Server processMessage had
UnknownHostException:"+uhe);
            }
      }
  }//processMessage()


  /**
   * Returns the number of Interfaces that have been stood up by this
ControlExecutive.
   * @return The number of Interfaces that have been stood up by this
ControlExecutive.
   */
  public int getNumberOfInterfaces(){
    return interfaceCount;
  }

  /**
   * Makes access to RoutingAlgorithm  possible for requiring classes.
   * @return RoutingAlgorithm object
   */
  public synchronized RoutingAlgorithm getRoutingAlgorithm(){
     return  routingAlgorithm;
  }

  /**
   * Makes access to AutoConfigurationExecutive  possible for requiring
classes.
   * @return AutoConfigurationExecutive  object
   */
   public AutoConfigurationExecutive getAutoConfigurationExecutive(){
     return  autoConExec;
  }

  /**
   * This is the method MessageProcessors use to register to process
Messages.
   * The ControlExecutive retrieves the list of Messages from mp by
calling
```

```
   * mp.getMessageTypes().
   * @param mp The MessageProcessor that is registering with this
ControlExecutive.
   */
  public void registerMessageProcessor(MessageProcessor mp){
      gui.sendText("Registering MessageProcessor: "+mp);
      Object[] elementsIProcess = null;
      //retrieve the list of Messages
      elementsIProcess = mp.getMessageTypes();
      for(int i=0;i<elementsIProcess.length;i++){
        String element = (String)elementsIProcess[i];
        if(!element.equals("saam.message.FlowResponse")){
          MessageProcessor oldProcessor =
            (MessageProcessor)messageProcessors.put(element,mp);
         // notify old Processor that it will no longer
         // receive this type of message.
        }else{
          if(mp.equals(this)){
            messageProcessors.put(element,this);
          }else{
            gui.sendText("DENIED: "+element);
          }
        }
      }
  }
  //henry
  public void registerMessageProcessor(Server server, MessageProcessor
mp){
    this.server = server;
    registerMessageProcessor(mp);
  }


  /**
   * ResidentAgentCustomers use this method to register to receive
ResidentAgent
   * updates from the ControlExecutive when they arrive.  If an agent
is replaced
   * with a new agent, the ControlExecutive will call the customer's
replaceAgent
   * method.
   * @param rac the ResidentAgentCustomer requesting registration.
   */
  public void registerCustomer(ResidentAgentCustomer rac){
      Object[] agentsIUse = null;
      agentsIUse = rac.getAgentTypes();
      for(int i=0;i<agentsIUse.length;i++){
        String agent = (String)agentsIUse[i];
        Vector customers = null;
        synchronized(agentCustomers){
          customers = (Vector)agentCustomers.get(agent);
        }
        if(customers == null){
          customers = new Vector();
          agentCustomers.put(agent,customers);
        }
        customers.add(rac);
```

```
        // oldProcessor.
      }
    }

    /**
     * Replaces an existing ResidentAgent with an incoming ResidentAgent
     * of the same class name.  If there are multiple instances of the
existing
     * agent, the replacement will occur instance for instance.
     * @param classObject The Class of the incoming agent.
     * @param className The class name of the incoming ResidentAgent
subclass
     */
    private void replaceOldAgent(Class classObject, String className){

      boolean badAgent = false;
      Vector agentInstances = new Vector();
      int numberOfInstancesNeeded = 1;
      if(className.equals("saam.residentagent.router.Scheduler")){
        synchronized(interfaces){
          numberOfInstancesNeeded = interfaces.size();
        }
      }
      for (int i=0;i<numberOfInstancesNeeded;i++){
        try{
          gui.sendText("About to install new agent");
          ResidentAgent newAgent = (ResidentAgent)
            classObject.newInstance();
          newAgent.install(this);
          gui.sendText("New agent installed");
          agentInstances.add(newAgent);
          gui.sendText("Instances present: "+agentInstances.toString());
          numberOfSchedulersPresent++;
        }catch(Exception e){
          gui.sendText("ResidentAgent bad: "+e.toString());
          badAgent = true;
        }
      }
      if(!badAgent){
        gui.sendText("Agent "+
          (numberOfInstancesNeeded==0? "not instantiated.":"instantiated
"+
          (numberOfInstancesNeeded==1? "once.": numberOfInstancesNeeded+"
times")));

        boolean agentAlreadyInstalled = false;
        synchronized(agents){
          agentAlreadyInstalled=agents.containsKey(className);
        }
        if(agentAlreadyInstalled){
          gui.sendText(className+" already resident");
          Vector previousAgents = (Vector)agents.remove(className);
          gui.sendText("Uninstalling previous agent: ");
          gui.sendText("Removing from channels...");
          for(int i=0;i<previousAgents.size();i++){
            ResidentAgent previousAgent = (ResidentAgent)
              previousAgents.get(i);
```

```
        if(previousAgent instanceof SaamTalker){
          removeTalkerFromAllChannels(
              (SaamTalker)previousAgent);
        }
        //every ResidentAgent is a SaamListener
        removeListenerFromAllChannels(
            previousAgent);
        ResidentAgent replacement = (ResidentAgent)
          agentInstances.get(i);
        if(previousAgent!=null){
          gui.sendText("Previous agent uninstalling");
          previousAgent.transferState(replacement);
          previousAgent.uninstall();
          previousAgent = null;
        }else{
          gui.sendText("No previous agent installed");
        }
      }
    }
    for (int i=0;i<numberOfInstancesNeeded;i++){
      ResidentAgent replacement = (ResidentAgent)
        agentInstances.get(i);
      notifyAgentCustomers(replacement, className);
      gui.sendText("Notifying customers, agent: "+replacement);
    }
    agents.put(className,agentInstances);
  }
}

/**
 * Here, the ControlExecutive iterates through the Vector of
 * ResidentAgentCustomers and calls the replaceAgent method of
 * each customer, passing the new agent.
 */
private void notifyAgentCustomers(
  ResidentAgent ra, String className){
    Vector customersOfThisAgent = (Vector)
      agentCustomers.get(className);
    if(customersOfThisAgent!=null){
      for(int i=0;i<customersOfThisAgent.size();i++){
        ((ResidentAgentCustomer)
          customersOfThisAgent.get(i)).
            replaceAgent(ra);
      }
      gui.sendText("Agent replaced: "+ra);
      gui.sendText("Customers: "+customersOfThisAgent);

    }
}

/**
 * In this method we would reflect into the Class Object and perform
 * a series of policy-related checks to determine whether or not the
 * agent is safe to instantiate.
 */
private boolean examineAgent(Class classObject){
  //future work..
```

```
    return true;
  }

  /**
   * This method is called by the Channels this Object has registered
to
   * monitor when a talker sends events on those Channels.
   * @param se The SaamEvent to be communicated.
   */
  public void receiveEvent(SaamEvent se){

    //the ControlExecutive only listens on the Channel between itself
and
    //the PacketFactory. Two types of traffic are sent on this Channel,
    //ResidentAgentEvents and MessageEvents

    if(se instanceof ResidentAgentEvent){
      Class classObject = ((ResidentAgentEvent)se).
        getClassObject();
      String className = classObject.getName();
      gui.sendText("received residentagent : "+className);
      //16 Jan 2000 akkoc added to determine the serveragent installed

if(className.equals("saam.residentagent.server.ServerAgentSymetric")){
        isServer = true;
      }
      if (examineAgent(classObject)){
        gui.sendText("replacing old residentagent....");
        replaceOldAgent(classObject, className);
      }else{
        //notify someone that the agent failed the inspection
      }

    }else if (se instanceof MessageEvent){

      MessageEvent me = (MessageEvent)se;
      String name =  me.getMessage().getClass().getName();
      gui.sendText("\nreceived messageevent : "+name);
     // System.out.println("received messageevent : "+name);

      //call the appropriate MessageProcessor to handle this Message
      MessageProcessor mp = null;
      mp = (MessageProcessor) messageProcessors.get(name);

      try{
        gui.sendText(" Calling Processor: "+mp.getClass().toString());
      //  System.out.println("\nCalling Processor:
"+mp.getClass().toString());
        mp.processMessage(me.getMessage());
      }catch(NullPointerException npe){
        //notify the sender that we do not have a
        //processor that is capable of processing this
        //Message.
        gui.sendText(" No Processor Available for " + name);
      }//try-catch
    }//not a ResidentAgentEvent or a MessageEvent
    mainGui.updateDisplay();
```

```
  }//receiveEvent()

  /**
   * Returns the Vector of Interfaces that have been instantiated by
this
   * ControlExecutive.
   * @return The Vector of Interfaces that have been instantiated by
this
   * ControlExecutive.
   */
  public Vector getInterfaces(){
      return interfaces;
  }//getInterfaces

  /**
   * The order in which Interfaces are instantiated is preserved.  Here
   * an Object can retrieve a specific Interface by instance number.
   * @param interfaceNumber The instance number of the Interface to be
   * retrieved.
   * @return The nth instance of Interface where n = interfaceNumber.
   */
  public Interface getInterface(int interfaceNumber){
      return (Interface)interfaces.get(interfaceNumber);
  }

  /**
   * Returns the Vector of InterfaceIDs assigned to the Interfaces
   * instantiated by this ControlExecutive.
   * @return The Vector of InterfaceIDs assigned to the Interfaces
   * instantiated by this ControlExecutive.
   */
  public Vector getInterfaceIDs(){
      return interfaceIDs;
  }//getInterfaces

  /**
   * Returns the Enumeration of Channels that have been instantiated
   * by this ControlExecutive.
   * @return The Enumeration of Channels that have been instantiated
   * by this ControlExecutive.
   */
  public Enumeration getActiveChannels(){
    return activeChannels.elements();
  }//getActiveChannels()

  /**
   * Returns true if the Channel has been instantiated by this
ControlExecutive.
   * @return True if the Channel has been instantiated by this
ControlExecutive.
   */
  public boolean isActiveChannel(int channel_ID){
    return activeChannels.containsKey(new Integer(channel_ID));
  }

  /**
```

```
   * To determine whether or not a talker is allowed to talk.  If
   * this method returns false, the talker will not be able to talk
   * on any Channels.
   * @param talker The SaamTalker to be verified.
   * @return True if the SaamTalker is allowed to talk.
   */
  private boolean verifyTalker(SaamTalker talker){
    return true;
  }//verifyRequestor()

  /**
   * To determine whether or not a listener has access to a given
Channel.
   * This method would be used to implement policy issues related to
access
   * control.
   * @param listener The listener to be verified.
   * @param channel_ID The ID of the Channel.
   */
  private boolean verifyChannelAccess(
    SaamListener pl, int channel_ID){
    //here, we would set the policy for channel_ID access.
    //i.e. we can restrict access of certain channel_ID to a
    //select list of listeners.  maybe a Hashtable called
    //"authorizationTable" which contains a Vector of
    //"authorizedListeners" and is keyed on channel_ID.
    return true;
  }//verifyAccess()

  /**
   * To determine whether or not a talker has access to a given
Channel.
   * This method would be used to implement policy issues related to
access
   * control.
   * @param talker The talker to be verified.
   * @param channel_ID The ID of the Channel.
   */
  private boolean verifyChannelAccess(
    SaamTalker talker, int channel_ID){
    //here, we would set the policy for channel_ID access.
    //i.e. we can restrict access of certain channel_ID to a
    //select list of listeners.  maybe a Hashtable called
    //"authorizationTable" which contains a Vector of
    //"authorizedListeners" and is keyed on channel_ID.
    return true;
  }//verifyAccess()

  /**
   * As the name implies, this method removes talker from the talker
   * Vectors of all Channels it has registered to talk on.
   * @param talker The talker to be removed.
   */
  public void removeTalkerFromAllChannels(SaamTalker talker){
    if(channelsTalkerHas.containsKey(talker)){
      Vector channels = null;
      synchronized(channelsTalkerHas){
```

```
        channels =(Vector)channelsTalkerHas.get(talker);
      }
      Enumeration e = ((Vector)channelsTalkerHas.get(talker)).
        elements();
      while(e.hasMoreElements()){
        Channel thisChannel = (Channel)e.nextElement();
        thisChannel.removeTalker(talker);
        gui.sendText(talker.toString()+
          " removed from channel "+
          thisChannel.getChannel_ID());
        gui.sendText("The Vector: "+channels.toString());
      }
      channelsTalkerHas.remove(talker);
    }
  }

  /**
   * As the name implies, this method removes listener from the
listener
   * Vectors of all Channels it has registered to listen on.
   * @param listener The listener to be removed.
   */
  public void removeListenerFromAllChannels(
    SaamListener listener){

    if(channelsListenerHas.containsKey(listener)){
      Vector channels = null;
      synchronized(channelsListenerHas){
        channels = (Vector)channelsListenerHas.get(listener);
      }
      Enumeration e = channels.elements();
      while(e.hasMoreElements()){
        Channel thisChannel = (Channel)e.nextElement();
        thisChannel.removeListener(listener);
        gui.sendText(listener.toString()+
          " removed from channel "+
          thisChannel.getChannel_ID());
        gui.sendText("The Vector: "+channels.toString());
      }
      channelsListenerHas.remove(listener);
    }
  }

  /**
   * SaamTalkers use this method to attach themselves to a Channel.  If
this
   * method succeeds, the talker will be allowed to transmit events on
this
   * Channel.
   * @param talker The talker requesting permission to talk on a
Channel.
   * @param channel_ID The ID of the channel to be utilized.
   */
  public void addTalkerToChannel(SaamTalker talker, int channel_ID)
    throws ChannelException {

      if(!verifyTalker(talker)){
```

```
        throw new ChannelException("Talking Denied");
      }

      //now test to see whether this channel_ID is within the
      //range of channel_IDs on which this requestor is authorized
      //to talk (policy issue).
      if (!verifyChannelAccess(talker, channel_ID)){
          throw new ChannelException("Access Denied");
      }
      Channel channel = null;
      synchronized(activeChannels){
        channel = (Channel)activeChannels.get(new Integer(channel_ID));
      }
      if(channel==null) {
        channel = new Channel(channel_ID,talker);
      }
      activeChannels.put(new Integer(channel_ID),channel);
        channel.addTalker(talker);
        mainGui.updateDisplay();
      if(channelsTalkerHas.containsKey(talker)){
        synchronized(channelsTalkerHas){
          ((Vector)channelsTalkerHas.get(talker)).
            add(channel);
        }
      }else{
        Vector vectorOfChannels = new Vector();
        vectorOfChannels.add(channel);
        channelsTalkerHas.put(talker, vectorOfChannels);
      }
//      gui.sendText("Talker added:");
//      gui.sendText(channel.toString());
  }//addTalkerToChannel()

  /**
   * Allows a SaamListener to monitor an emulated UDP port
   * @param listener The listener requesting to monitor a port.
   * @param port The port to be monitored.
   */
  public void monitorPort(SaamListener listener, int port)
    throws PortAccessDeniedException{
    //presumably, the listener has already been verified
    //by the Control Executive and placed on an access
    //list within the EventController.  There is no such
    //access list at this time.
      try{
        if(!hasListener(port)){
         addTalkerToChannel(transportInterface,port);
         addListenerToChannel(listener, port);
//         gui.sendText(listener.toString()+
//           " listening to port: "+port);
        }else {
          gui.sendText(listener.toString()+
            " denied access to port: "+port);
          throw new PortAccessDeniedException("Port in use");
        }
      }catch(ChannelException ce){
        throw new PortAccessDeniedException("Not authorized");
```

```
      }//try-catch
  }

  /**
   * SaamListeners use this method to attach themselves to a Channel.
If this
   * method succeeds, the listener will receive all events that are
sent on this
   * Channel.
   * @param listener The listener requesting to monitor a Channel.
   * @param channel_ID The ID of the channel to be monitored.
   */
  public void addListenerToChannel(
    SaamListener listener, int channel_ID)
    throws ChannelException{

    if (verifyChannelAccess(listener,channel_ID)){

      Channel channel = null;
      synchronized(activeChannels){
        channel = (Channel)activeChannels.get(new Integer(channel_ID));
      }
      if(channel==null) {
        channel = new Channel(channel_ID,listener);
      }

      //no effect if the Channel is already on the active list
      activeChannels.put(new Integer(channel_ID),channel);
      channel.addListener(listener);
//      gui.sendText("Listener added:");
//      gui.sendText(channel.toString());
      if(channelsListenerHas.containsKey(listener)){
        synchronized(channelsListenerHas){
          ((Vector)channelsListenerHas.get(listener)).
            add(channel);
        }
      }else{
        Vector vectorOfChannels = new Vector();
        vectorOfChannels.add(channel);
        channelsListenerHas.put(listener, vectorOfChannels);
      }

    }//if
  }//addListenerToChannel()

  /**
   * Used to determine if any Objects are registered to listen on the
Channel
   * with channel_ID.
   * @param channel_ID The ID of the Channel to be queried.
   */
  public boolean hasListener(int channel_ID){
    try{
      Channel channel = null;
      synchronized(activeChannels){
        channel = (Channel)activeChannels.get(new Integer(channel_ID));
      }
```

```
      return channel.hasListeners();
    }catch(NullPointerException npe){}
      return (false);

  }//hasListener()

  /**
   * Used to determine if any Objects are registered to talk on the
Channel
   * with channel_ID.
   * @param channel_ID The ID of the Channel to be queried.
   */
  public boolean hasTalker(int channel_ID){
    try{
      Channel channel = null;
      synchronized(activeChannels){
        channel = (Channel)activeChannels.get(new Integer(channel_ID));
      }
      return channel.hasTalkers();
    }catch(NullPointerException npe){}
      return (false);

  }//hasListener()

  /**
   * Once approved to communicate on a channel, a
   * SaamTalker calls this method to actually broadcast
   * events on the channel.  A ChannelException will be
   * thrown if the talker is not registered to talk on
   * the channel contained in the SaamEvent.
   */
  public void talk(SaamEvent event) throws
    ChannelException{
     // System.out.println("INSIDE TALK ...");
      SaamTalker talker = event.getTalker();
      int channel_ID = event.getChannel_ID();
      Channel channel = null;
//    synchronized(activeChannels){
      synchronized(theLock) {
// Questions to Dean:
// (1) Is the above sufficient?
// (2) Does this support talking to multiple channels?
// (3) Why are more and more ">>> Ready to ..." msgs printed out?

      channel = (Channel)activeChannels.get(new
Integer(event.getChannel_ID()));
      if (channel == null) {
        gui.sendText(talker.toString()+">>> has no channel.to talk");
        return;
      }
      // }// old LOCK ENDS HERE
      if(channel.isRegistered(talker)){
        //order the channel to notify its listeners
        gui.sendText(talker.toString()+">>> is Ready to channel.talk");
        channel.talk(event);
        channel.setTimeLastUsed();
      }else{
```

```
        gui.sendText("ACCESS DENIED! Unregistered talker: "+
talker.toString() +"\n"+
                                    "Attempted to talk on channel
"+channel_ID);
        throw new ChannelException(
           talker.toString()+" not Registered on "+
           "channel "+channel_ID+".");
       }
      }// new LOCK ens here
  }//talk()

  /**
   * Displays the status of all Channels that have been instantiated by
the
   * ControlExecutive.  Channels are displayed in the
ControlExecutive's gui.
   * @param msg The text to appear before the channels are displayed.
   */
  public void displayActiveChannels(String msg){

/*      gui.sendText("\n"+msg);
      gui.sendText("Active channels:");
      Enumeration e = activeChannels.keys();
      while(e.hasMoreElements()){
        Integer key = (Integer)(e.nextElement());
        Channel channel =
          (Channel)activeChannels.get(key);
        gui.sendText(channel.toString());
      }//while(e.hasMoreElements())
*/
  }//displayActiveChannels()

  /**
   * Removes a SaamListener from the Vector of listeners associated
with the
   * Channel containing channel_ID
   * @param sl The SaamListener to be removed.
   * @param channel_ID The ID of the desired Channel.
   */
  public void removeListenerFromChannel(
    SaamListener sl, int channel_ID){

      Channel channel = null;
      synchronized(activeChannels){
         channel = (Channel)
          activeChannels.get(new Integer(channel_ID));
      }
      channel.removeListener(sl);
  }//closeChannelConnection()

  /**
   * Returns a <code>String</code> representation of this object
   * @return The <code>String</code> representation of this object
   */
  public String toString(){
    return ("Control Executive");
  }
```

```
} //end of class CONTROL EXECUTIVE
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX L – SAAM CONTROL.PACKETFACTORY CLASS CODE

```
//24Feb2000[Henry]            - modified
// Feb 2000[akkoc]    - modified
// 01Aug99 [Vrable]   - Created
package saam.control;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Enumeration;
import java.util.Vector;
import java.util.TooManyListenersException;
import java.util.StringTokenizer;
import java.lang.reflect.Constructor;
import java.net.UnknownHostException;

import saam.net.*;
import saam.event.*;
import saam.message.*;
import saam.util.*;
import saam.residentagent.*;

/**
* A PacketFactory can be used to build SaamPackets for sending or
* to receive SaamPackets and extract their atomic elements.  These
* atomic elements are currently one of two types:  A subclass of
* saam.residentagent.ResidentAgent or a subclass of
* saam.message.Message.<p>
* A sender would instantiate a PacketFactory to build
* Saam Packets.  The PacketFactory's append methods receive
* Message Objects, ResidentAgent Objects, or a String that represents
* the class name of a ResidentAgent as parameters and then dynamically
* construct the appropriate header based on the number of elements
* received and the current time.  The getBytes method is used to
* retrieve the byte array that represents the SAAMPacket that has been
* constructed by this PacketFactory.<p>
* The ControlExecutive uses the PacketFactory to receive and parse
* SaamPackets.
*/
public class PacketFactory extends Thread
  implements SaamTalker, SaamListener{

  private final boolean guiActive = true;
  private SAAMRouterGui gui;
  private ControlExecutive controlExec;
  private boolean started = false;
  private boolean firstEvent = true;
  private boolean bytesRetrieved;
  private byte[] packet,DCMpacket,PNpacket,UCMpacket;
  private byte numberOfMessages;
  private Loader loader;
  private Class message;
  private SaamEvent currentEvent;
```

```
private Thread owner;
private static int instanceNumber;
private Object theLock = new Object();

// xie
private FIFOQueue inputQueue = new FIFOQueue(1000);


/**
 * Use the no-args constructor to begin constructing packets
 * on the sending side.
 */
//no-args constructor doesn't come for free when we have
//another constructor
public PacketFactory(){
  //instanceNumber++;
  gui = new SAAMRouterGui("Outbound.."+toString());
  gui.setTextField("I construct outbound packets");
}

/**
 * This constructor is not available to Objects outside the
 * saam.control package.  The ControlExecutive uses this constructor
 * to receive and parse SAAMPackets.  The PacketFactory passes the
 * atomic elements (either ResidentAgents or Messages) up to the
 * ControlExecutive for further processing.
 * @param controlExec The ControlExecutive that is to receive
 * updates from this PacketFactory.
 */
PacketFactory(ControlExecutive controlExec){
  //this();
  gui=new SAAMRouterGui("Input.."+toString());
  gui.setTextField("I Listen for inbound packets");
  this.controlExec=controlExec;
  loader = new Loader();

  //*****************************
  //**Listen to desired Channels**
  //*****************************
  int channel_ID =
    ProtocolStackEvent.PACKETFACTORY_CHANNEL;
  try{
    controlExec.addListenerToChannel(this, channel_ID);
    gui.sendText("Listening to channel: "+channel_ID);
  }catch(ChannelException ce){
    gui.sendText(ce.toString());
  }//try-catch

  //***************************************
  //**Register to talk on desired Channels**
  //***************************************
  channel_ID = ControlExecutive.SAAM_CONTROL_PORT;
  try{
     controlExec.addTalkerToChannel(this,
       channel_ID);
     gui.sendText("Talking enabled on channel: " + channel_ID);
  }catch(ChannelException ce){
```

```
    gui.sendText(ce.toString());
   }

   //****** Huseyin UYSAL ***************************************/
   channel_ID = ProtocolStackEvent.FROM_PACKETFACTORY_TO_ACE;
   try{
     controlExec.addTalkerToChannel(this,channel_ID);
     gui.sendText("Talking enabled on channel: " + channel_ID);
   }catch(ChannelException che){
     gui.sendText(che.toString());
   }
   start();
}

/**
 * When instantiated to receive packets, the PacketFactory
 * Thread waits until a SAAMPacket arrives, then it calls
 * the processPacket method.
 *
public void run(){
   while(true){
      try{
        if(!started){
          synchronized(theLock){
            gui.sendText("Waiting...");
            while(!started) theLock.wait();
            started=true;
          }
        }
      }catch(InterruptedException ie){
        gui.sendText(ie.toString());
      }
      gui.sendText("Resumed");
      processPacket();
   }//while(started)
}
*/

/**
 * When instantiated to receive packets, the PacketFactory
 * Thread waits until a SAAMPacket arrives, then it calls
 * the processPacket method.
 */
public void run(){
   while(true){
     gui.sendText("\n Inside PacketFactory run()");
     synchronized (theLock){
       if (inputQueue.isEmpty()){
         started = false;
         try{
           gui.sendText("Waiting...");
           theLock.wait();
           gui.sendText("Continuing");
         }
         catch(InterruptedException e){
           gui.sendText("Interrupted exception catched");
         }
```

```
        }// end if

        packet = (byte[]) inputQueue.dequeue();
      }// end synchronization
      processPacket();
    }//while(true)
  }

  /**
   * This method is called by the Channels this Object has registered
to
   * monitor when a talker sends events on those Channels.
   * @param se The SaamEvent to be communicated.

  public synchronized void receiveEvent(SaamEvent se){
 *
  public  void receiveEvent(SaamEvent se){

    gui.sendText("\nGot a packet");
    currentEvent=se;
    //check to see if the currentThread has an owner, if it
    //does, notify the owner that the event has arrived.
    //otherwise, just process the packet.
    if(!firstEvent){
      synchronized(theLock){
        theLock.notify();
      }
      if(!started){

        synchronized(theLock){
          started=true;
          theLock.notify();
        }

      }else{
        processPacket();
      }
    }else{
      firstEvent=false;
      started=true;
      start();
    }
    se=null;
  }
  */

  /**
   * This method is called by the Channels this Object has registered
to
   * monitor when a talker sends events on those Channels.
   * @param se The SaamEvent to be communicated.

  public synchronized void receiveEvent(SaamEvent se){
 */
  public  void receiveEvent(SaamEvent se){

    gui.sendText("\n---- Got a packet");
```

```
    currentEvent = se;
    ProtocolStackEvent psec = (ProtocolStackEvent)currentEvent;

    byte[] newcomer = psec.getPacket();
    gui.sendText("\n New packet has length = " + newcomer.length);

    synchronized(theLock){
      inputQueue.enqueue((Object) newcomer);
      if (!started){
        started = true;
        gui.sendText("\n Waking up the processPacket thread");
        theLock.notify();
      }// end if
    }
  }

  /**
   * This method is used to extract the individual Class
   * Objects that are represented in the packet.  These Class
   * Objects are either of type 0 (ResidentAgent) or 1 (Message).<p>
   * If a ResidentAgent is received, a Class Object is created
   * that represents the agent.  That Class Object is then sent to
   * the ControlExecutive for screening and agent instantiation.<p>
   * If a Message is received, that Message is instantiated and sent
   * to the ControlExecutive for further processing.
   */
  private void processPacket() {
    int channel = currentEvent.getChannel_ID();
    String eventSource = (String)currentEvent.getSource();

    //packet is a byte array
    packet = ((ProtocolStackEvent)currentEvent).getPacket();


    //see saam.util for PrimitiveConversions and Array classes
    long timeStamp = PrimitiveConversions.getLong(
        Array.getSubArray(packet,0,8));
    numberOfMessages=packet[8];
    gui.sendText("packet arrived: " +
      "\n  source:       " + eventSource +
      "\n  channel:      " + channel +
      "\n  size:         " + packet.length +
      "\n  # of Messages: " + numberOfMessages +
      "\n  timeStamp:    " + timeStamp);

    //now we trim the packet by removing the header.
    packet = Array.getSubArray(packet,9,packet.length);

    //used to track the current position in the array.
    int index = 0;
            short length = 0;
            String elementName = "";

    //extract and process each atomic element of the packet
    //separately.  Here we assume the packet is a properly
    //formatted SAAMPacket when it arrives, and that the
    //length is less than the max allowed.
```

```
    for(int i=1;i<=numberOfMessages;i++){

      gui.sendText("\nProcessing Element["+i+"]:");
      byte type = packet[index++];
      gui.sendText(" type:  "+type);


      byte[] bytes;


      switch(type){
        case Message.RESIDENT_AGENT:
        case Message.MESSAGE_DEFAULT_TYPE:
        case Message.FAILURE:

          //retrieve the number of bytes the class name occupies
              byte nameLength = packet[index++];

              //extract the name of the class file as a byte array
              byte[] elementNameArray = Array.getSubArray(
                packet,index, index+nameLength);
              index+=nameLength;

              //convert the name back into a String
              elementName = new String(elementNameArray);
              gui.sendText("  Name: "+elementName);


          //retrieve the length of the Object
            length = PrimitiveConversions.getShort(
                  Array.getSubArray(packet,index,index+2));
              index+=2;
          gui.sendText("  Length:       "+length);
          bytes = Array.getSubArray(packet,index,index+length);
          index+=length;

          if(type==Message.RESIDENT_AGENT){
            gui.sendText("This is a ResidentAgent");
            //Assume this class is of type ResidentAgent
            try{
              //Attempt to define the class using the current
              //class loader.
              loader.defClass(elementName, bytes);
            }catch(LinkageError le){
              //If the loader already has a definition for the class
              //a LinkageError will be thrown.  If this happens, we
              //need to instantiate a new class loader and use it to
              //define the class.  A nice little trick we learned from
              //page 55 of Jason Hunter's "Java Servlet Programming"
book.
              gui.sendText(le.toString());
              gui.sendText("Class was previously loaded...");
              gui.sendText("Replacing old ClassLoader...");
              Loader newLoader = new Loader();
              newLoader.defClass(elementName, bytes);
            }
            try{
```

```
      //message is of type Class.
      message = Class.forName(elementName, true, loader);
    }catch(ClassNotFoundException cnfe){
      gui.sendText(cnfe.toString());
    }
    gui.sendText(message.toString());
    ResidentAgentEvent rae = new ResidentAgentEvent(
      eventSource,
      this,
      ControlExecutive.SAAM_CONTROL_PORT,
      message);
    try{
      gui.sendText("Forwarding on channel "+
        ControlExecutive.SAAM_CONTROL_PORT);
      controlExec.talk(rae);
    }catch(ChannelException tde){
      gui.sendText(tde.toString());
    }
  }//if ResidentAgent
  else if(type==Message.MESSAGE_DEFAULT_TYPE){

    gui.sendText("This is a Message");
    //Assume this class is of type Message.
    try{
      //message is of type Class.
      message = Class.forName(elementName);
    }catch(ClassNotFoundException cnfe){
      {gui.sendText("Bytecode for: "+elementName+
        " not found.");
      }
    }

    try{
      //Call the constructor from within this Class that
      //takes a byte array as its only argument
      Constructor cons = message.getConstructor(
        new Class[] {byte[].class});

      //Create the instance of this Message
      Message instance =
        (Message)cons.newInstance(
          new Object[] {bytes});
      gui.sendText(instance.toString());
      MessageEvent me = new MessageEvent(
        eventSource,
        this,
        ControlExecutive.SAAM_CONTROL_PORT,
        instance);
      //send this MessageEvent on the Control port.
      try{
        gui.sendText("Forwarding on channel "+
          ControlExecutive.SAAM_CONTROL_PORT);
        controlExec.talk(me);
      }catch(ChannelException tde){
        gui.sendText(tde.toString());
      }
    }catch(Exception e){
```

```
              //need to notify sender that we have no classfile
              //with this name
              gui.sendText(e.toString());
            }//try-catch
          }//else if default message type '1'

/*************************************************************/
          //Hasan UYSAL
          else if(type==Message.FAILURE){
            gui.sendText("This is an InterfaceFailure message
Arrived.");

            InterfaceFailure failure=new InterfaceFailure(bytes);
            MessageEvent failMes=new MessageEvent(
              toString(),
              this,ControlExecutive.SAAM_CONTROL_PORT,
              failure);

            try{
              controlExec.talk(failMes);
            }catch(Exception ex){
              gui.sendText("Problem with talking failure message.");
              continue;
            }
          }
          break;
/*********************************************************************/
        //Henry
        case Message.FLOWREQUEST_TYPE:
        case Message.FLOWRESPONSE_TYPE:
        case Message.RESOURCEALLOCATION_TYPE:
        case Message.SLSTABLEENTRY_TYPE:

          //retrieve the length of the Object
            length = PrimitiveConversions.getShort(
              Array.getSubArray(packet,index,index+2));
              index+=2;
          gui.sendText("  Length:       "+length);
          bytes = Array.getSubArray(packet,index,index+length);
          index+=length;

          if(type==Message.FLOWREQUEST_TYPE){
            gui.sendText("This is a FlowRequest Message");
            //processMessage(bytes, eventSource, "FlowRequest");
            controlExec.processMessage(bytes, "FlowRequest");
          }//flow request

          else if(type== Message.FLOWRESPONSE_TYPE){
            gui.sendText("This is a FlowResponse Message");
                  controlExec.processMessage(bytes, "FlowResponse");
            //processMessage(bytes, "FlowResponse");
          }//flow response

          else if(type==Message.RESOURCEALLOCATION_TYPE){
            gui.sendText("This is a ResourceAllocation Message");
            controlExec.processMessage(bytes, "ResourceAllocation");
          }// resource allocation
```

```
          else if(type== Message.SLSTABLEENTRY_TYPE){
            gui.sendText("This is a SLSTableEntry Message");
            if (bytes.length == SLSTableEntry.REMOVE_SLS_TYPE) {
                  controlExec.processMessage(bytes, "SLSTableEntry");
            }
            else {
                  processMessage(bytes, eventSource, "SLSTableEntry");
            }
          }//slstableentry
          break;

        case Message.FLOWTERMINATION_TYPE:
          length = 4;
          gui.sendText("  Length:       "+length);
          bytes = Array.getSubArray(packet,index,index+length);
          index+=length;
              gui.sendText("This is a FlowTermination Message");
              controlExec.processMessage(bytes, "FlowTermination");
          break;

/*********************************************************************/

        //Hasan AKKOC
        case Message.DCM_TYPE:
        case Message.PARENT_NOTIFICATION_TYPE:

          if(type==Message.DCM_TYPE){
            gui.sendText("This is a DCM  Message");

            try{
              DCM  dcm = new DCM(packet);
              gui.sendText("DCM message ia created.");
              gui.sendText(dcm.toString());
              MessageEvent me = new MessageEvent( eventSource, this,

ControlExecutive.SAAM_CONTROL_PORT,dcm);
              //send this MessageEvent on the Control port.
              try{
                gui.sendText("Forwarding on channel "+
                      ControlExecutive.SAAM_CONTROL_PORT);
                controlExec.talk(me);
                gui.sendText("DCM is sent to ControlExecutive.");
              }catch(ChannelException tde){
                gui.sendText(tde.toString());
              }
            }catch(Exception e){
              gui.sendText(e.toString());
            }//try-catch
          }//DCM

          else if(type== Message.PARENT_NOTIFICATION_TYPE){
            gui.sendText("This is a ParentNotification Message");
            //Assume this class is of type Message.
            try{
              ParentNotification pn = new ParentNotification(packet);
              MessageEvent me = new MessageEvent( eventSource, this,
```

```
                                        ControlExecutive.SAAM_CONTROL_PORT,pn);
                    //send this MessageEvent on the Control port.
                    try{
                      gui.sendText("Forwarding on channel "+
                      ControlExecutive.SAAM_CONTROL_PORT);
                      controlExec.talk(me);
                    }catch(ChannelException tde){
                      gui.sendText(tde.toString());
                    }
                  }catch(Exception e){
                    gui.sendText(e.toString());
                  }//try-catch
                }//PARENT notification
                break;


/*********************************************************************/
//modified by Huseyin UYSAL

          case Message.UCM_TYPE:
            gui.sendText("This is a UCM  Message");
            int flowIdOfServer=PrimitiveConversions.getInt(
                Array.getSubArray(packet,index,index+4))-1;
            gui.sendText("Flow id of the sever is "+flowIdOfServer);
            if(!controlExec.getIsServer()){
              //I need UCM and the LSAs as a total
              // need to pass the numberOfMessages to UCM handler
              gui.sendText("This is a router so forwarding the packet to
ACE");
              //byte [] tem = Array.concat(numberOfMessages,type);
              //packet=Array.concat(type,packet);
              packet = Array.concat(numberOfMessages,packet);
              i=numberOfMessages;
              int channelToAce =
ProtocolStackEvent.FROM_PACKETFACTORY_TO_ACE;
              //create a ProtocolStackEvent and send it to ACE
              ProtocolStackEvent stackEvent=new ProtocolStackEvent(
                this.toString(),
                this,
                channelToAce,
                packet);

              try{
                gui.sendText("Forwarding on channel "+channelToAce);
                controlExec.talk(stackEvent);
              }catch(ChannelException tde){
                gui.sendText(tde.toString());
              }
            }
            else{
              //this is the server that UCM is destined
              int numberOfRoutersInUCM=PrimitiveConversions.getInt(
                  Array.getSubArray(packet,index+20,index+24));
              gui.sendText("Number of reachable routers is "
                  +numberOfRoutersInUCM);
              int UCMLength=4+16+4+4+numberOfRoutersInUCM*16;
              bytes=Array.getSubArray(packet,index,index+UCMLength);
```

```
            bytes=Array.concat(type,bytes);
            index+=UCMLength;
            try{
              UCM ucm  =  new UCM(bytes);
              gui.sendText("\n"+ucm.toString());
              MessageEvent me = new MessageEvent(
                eventSource,
                this,
                ControlExecutive.SAAM_CONTROL_PORT,
                ucm);
              gui.sendText("Forwarding packet on channel "+
                ControlExecutive.SAAM_CONTROL_PORT);
              controlExec.talk(me);
            }catch(Exception ex){
              gui.sendText(ex.toString());
            }
          }//else
          break;


/*****************************************************************/
     //Huseyin UYSAL
        //if it is link state Advertisement
        case Message.LSA:
          //i got a linkstate advertisement so I need to process
accordingly
          //processLSAMessage();
          gui.sendText("This is a Link State Advertisement");
          //Assume this class is of type Message.
          IPv6Address  router=null;
          try{
            router = new IPv6Address(Array.
              getSubArray(packet,index,index+IPv6Address.length));
          }catch(UnknownHostException hoe){
            gui.sendText("An exception occured while forming LSApacket
at PacketFactory");
          }
          LinkStateAdvertisement LSA = new
LinkStateAdvertisement(router);
          index+=IPv6Address.length;
          byte numberOfInterfaces=packet[index++];
          for(int ix=0;ix<numberOfInterfaces;ix++){
            byte mesType = packet[index++];
            IPv6Address interfaceIP=null;
            try{
              interfaceIP = new IPv6Address(Array.getSubArray(
                packet,index,index+IPv6Address.length));
            }catch(UnknownHostException ex){
              gui.sendText("Exception occured while forming LSA
packet");
            }
            index+=IPv6Address.length;
            int bandwidth = PrimitiveConversions.getInt(Array.
              getSubArray(packet,index,index+4));
            index+=4;
            byte numSLPs=packet[index++];
            InterfaceLSA tempLSA = new InterfaceLSA(
```

```
                    interfaceIP,bandwidth,mesType);
              Vector V=new Vector(4);
              for(int j=0;j<numSLPs;j++){
                SLPLSA slpLSA = new SLPLSA(Array.getSubArray(
                  packet,index,index+SLPLSA.length));
                V.add(slpLSA);
                index+=SLPLSA.length;
              }//end for SLPLSA creation
              tempLSA.insertSLP(V);
              LSA.insertInterfaceLSA(tempLSA);
            }//end for


            //I need to create a Protocol Stack event and sent this
            //to control exec first chech the lsa Type

            MessageEvent me = new MessageEvent(
              eventSource,this,ControlExecutive.SAAM_CONTROL_PORT,LSA);

            try{
              gui.sendText("Forwarding on channel "
                  + ControlExecutive.SAAM_CONTROL_PORT);
              controlExec.talk(me);
            }catch(Exception e){
              gui.sendText(e.toString());
            }

            break;
/*********************************************************************/
          default:
            gui.sendText("Packet type unrecognized: "+type);
            //packet type is unrecognized.  Here we could
            //extract a channel_ID that could be embedded
            //in the packet, and then send the unrecognized
            //element on that channel.
        }//end switch
      }//for

//    started=false;
 }//processPacket()

 //Henry
 private void processMessage(byte[] bytes,
       String eventSource, String messageType){

   //Assume this class is of type Message.
   try{
     //message is of type Class.
     message = Class.forName("saam.message."+messageType);
   }
   catch(ClassNotFoundException cnfe){
     {gui.sendText("Bytecode for: saam.message."
         +messageType+" not found.");
     }
   }

   try{
```

```
      //Call the constructor from within this Class that
      //takes a byte array as its only argument
      Constructor cons = message.getConstructor(
        new Class[] {byte[].class});

      //Create the instance of this Message
                  gui.sendText("Calling constructor:
"+cons.toString());
      Message instance =
        (Message)cons.newInstance(
          new Object[] {bytes});
      gui.sendText("Instance of message created:
"+instance.toString());
      MessageEvent me = new MessageEvent( eventSource, this,
              ControlExecutive.SAAM_CONTROL_PORT,instance);
                  gui.sendText(me.toString());
      //send this MessageEvent on the Control port.
      try{
        gui.sendText("Forwarding on channel "+
          ControlExecutive.SAAM_CONTROL_PORT);
        controlExec.talk(me);
      }catch(ChannelException tde){
        gui.sendText(tde.toString());
      }
    }catch(Exception e){
      //need to notify sender that we have no classfile
      //with this name
      gui.sendText("processMessage: "+e.toString());
    }//try-catch
  }

  //Henry
  private void processMessage(byte[] bytes, String messageType){

    //Assume this class is of type Message.
    try{
      //message is of type Class.
      message = Class.forName("saam.message."+messageType);
    }
    catch(ClassNotFoundException cnfe){
      {gui.sendText("Bytecode for: saam.message."
          +messageType+" not found.");
      }
    }

    try{
      //Call the constructor from within this Class that
      //takes a byte array as its only argument
      Constructor cons = message.getConstructor(
        new Class[] {byte[].class});

      //Create the instance of this Message
      Message instance =
        (Message)cons.newInstance(
          new Object[] {bytes});
      gui.sendText(instance.toString());
      controlExec.processMessage(instance);
```

```
    }catch(Exception e){
      //need to notify sender that we have no classfile
      //with this name
      gui.sendText("processMessage: "+e.toString());
    }//try-catch
  }

  /**
   * This method can be used to append a Message to an outgoing
   * SAAMPacket.  To later retrieve the entire packet (with header)
   * as a byte array, call the getBytes method.
   * @param me The Message to be appended.
   */
  public void append(Message me){
    if(bytesRetrieved){
      packet=null;
      numberOfMessages=0;
      bytesRetrieved = false;
    }
    byte type = me.getType();
    String name = me.getClass().getName();
    byte nameLength = (byte)name.getBytes().length;
    byte[] parameters = me.getBytes();

    //here we could check the length of the parameter array supplied
    //with the length returned from the length() method call.
    short paramLength = (short)parameters.length;
    gui.sendText("\nappending "+name+" with length = "+paramLength);
    //now append the Message to the packet byte array
    packet = Array.concat(packet,type);
    if (type <= 1) {
          packet = Array.concat(packet,nameLength);
          packet = Array.concat(packet,name.getBytes());
    }
    //new packet format only requires these
    if (type != (int)Message.FLOWTERMINATION_TYPE) {
      //message with variable length
          packet = Array.concat(packet,
            PrimitiveConversions.getBytes(paramLength));
    }
    packet = Array.concat(packet,parameters);

    //increment the count of messages in this packet
    numberOfMessages++;

    gui.sendText("Appended Message:" +
      "\n  Type:           " + type +
      "\n  name:           " + name +
      "\n  param length:   " + paramLength +
      "\n  # of messages:  " + numberOfMessages +
      "\n  packet length:  " + packet.length+"\n");

  }  //end of append

  /**For handling new SAAMPacket format
   * This method can be used to append a Message to an outgoing
   * SAAMPacket.  To later retrieve the entire packet (with header)
```

```
   * as a byte array, call the getBytes method.
   * @param me The Message to be appended.
   */
      public void append(byte[] message){
    if(bytesRetrieved){
      packet=null;
      numberOfMessages=0;
      bytesRetrieved = false;
    }

    packet = Array.concat( packet, message );
    numberOfMessages++;
    gui.sendText("Appended byte type message ");
  }

  /**
   * This method can be used to append a DCM message to an outgoing
   * SAAMPacket.  To later retrieve the entire packet (with header)
   * as a byte array, call the getDCMBytes method.
   * @param downWard The DCM message to be appended.
   */
  public void appendDCM( DCM downWard){
    gui.sendText(" Appending a dcm message before sending downward with
lengh");
    if(bytesRetrieved){
      DCMpacket=null;
      numberOfMessages=0;
      bytesRetrieved = false;
    }
    DCMpacket = Array.concat(DCMpacket,downWard.getBytes());
  }//end of appendDCm

  /**
   * This method can be used to append a PN message to an outgoing
   * SAAMPacket.  To later retrieve the entire packet (with header)
   * as a byte array, call the getPNBytes method.
   * @param downWard The PN message to be appended.
   */
  public void appendPN( ParentNotification pn){
    gui.sendText(" Appending a PN message before sending downward with
lengh");
    if(bytesRetrieved){
      PNpacket=null;
      numberOfMessages=0;
      bytesRetrieved = false;
    }
    PNpacket = Array.concat(PNpacket,pn.getBytes());
    gui.sendText("after appending PN is "+PNpacket.length);

  }//end of appendDCm

  /**
   * This method can be used to append a PN message to an outgoing
   * SAAMPacket.  To later retrieve the entire packet (with header)
   * as a byte array, call the getPNBytes method.
   * @param downWard The PN message to be appended.
   *
```

```
public void appendUCM( UCM upWard){
  gui.sendText(" Appending a UCM message before sending upward");
  UCMpacket = Array.concat(UCMpacket,upWard.getBytes());
}//end of appendUCM
*/

/**
 * added by Huseyin UYSAL
 *
 */
public void appendUCM(byte numMes,byte[] bytes){
  if(bytesRetrieved){
    packet=null;
    numberOfMessages=0;
    bytesRetrieved = false;
  }
  packet=Array.concat(packet,bytes);
  numberOfMessages=numMes;
}


/**
 * This method can be used to append a ResidentAgent to an outgoing
 * SAAMPacket.  To later retrieve the entire packet (with header)
 * as a byte array, call the getBytes method.
 * @param ra The ResidentAgent to be appended.
 */
public void append(ResidentAgent ra) throws IOException{
  String name = ra.getClass().getName();
  append(name);
}


/**
 * This method can be used to append a ResidentAgent by name to an
 * outgoing SAAMPacket.  To later retrieve the entire packet
 * (with header) as a byte array, call the getBytes method.
 * @param residentAgentClassName The String name of the ResidentAgent
 * classfile to be appended.
 */
public void append(String residentAgentClassName)
  throws IOException{
  if(bytesRetrieved){
    packet=null;
    numberOfMessages=0;
    bytesRetrieved = false;
  }
  byte type = 0;
  String name = residentAgentClassName;
  //String fileName = ".."+File.separatorChar +  //for KAWA
  String fileName = "..\\.."+File.separatorChar +    //for Jbuilder
    residentAgentClassName.replace('.',File.separatorChar);
  fileName+=".class";
  gui.sendText("File name: "+fileName);
  FileInputStream fis = null;
  try{
    fis = new FileInputStream(fileName);
```

```
    }catch(IOException ioe){
      throw new IOException(
        "Problem reading ResidentAgent: "+fileName);
    }
    byte nameLength = (byte)name.getBytes().length;
    byte[] byteCode = new byte[fis.available()];
    short length = (short)fis.read(byteCode);

    packet = Array.concat(packet,type);
    packet = Array.concat(packet,nameLength);
    packet = Array.concat(packet,name.getBytes());
    packet = Array.concat(packet,
      PrimitiveConversions.getBytes(length));
    packet = Array.concat(packet,byteCode);
    numberOfMessages++;

    gui.sendText("Appended ResidentAgent:" +
      "\n  Type:              " + type +
      "\n  name:              " + name +
      "\n  byteCode length:   " + length +
      "\n  # of messages:     " + numberOfMessages +
      "\n  packet length:     " + packet.length+"\n");

  }

  /**
   * Appends a header to the byte array.  The header conforms
   * to the structure of a SAAMHeader.
   */
  private void appendHeader(){
    byte[] timeStamp = PrimitiveConversions.getBytes(
      System.currentTimeMillis());
      packet = Array.concat(numberOfMessages,packet);
      packet = Array.concat(timeStamp,packet);
      gui.sendText("Appended header:"+
        "\n  timeStamp:    "+PrimitiveConversions.getLong(
          Array.getSubArray(packet,0,8))+
        "\n  # of updates: "+packet[8] +
        "\n  packet length: "+packet.length+"\n");
  }

  /**
   * Returns a byte array that conforms to the structure of
   * a SAAMPacket.
   * @return A byte array that conforms to the structure of
   * a SAAMPacket.
   */
  public byte[] getBytes(){
    appendHeader();
    bytesRetrieved = true;
    return packet;
  }

  /**
   * Returns a byte array that conforms to the structure of a
DCMPacket.
   * @return A byte array that conforms to the structure of DCMPacket.
```

```
    */

    public byte[] getDCMBytes(){
     bytesRetrieved = true;
     return DCMpacket;
   }

   /**
    * Returns a byte array that conforms to the structure of a PNPacket.
    * @return A byte array that conforms to the structure of PNPacket.
    */
   public byte[] getPNBytes(){
     bytesRetrieved = true;
     return PNpacket;
   }

   /**
    * Returns a byte array that conforms to the structure of a
UCMPacket.
    * @return A byte array that conforms to the structure of UCMPacket.
    */
   public byte[] getUCMBytes(){
     bytesRetrieved = true;
     return UCMpacket;
   }

   /**
    * Returns the current length of the packet.
    * @return The current length of the packet.
    */
   public int length(){
     try{
       return packet.length;
     }catch(NullPointerException npe){
       return 0;
     }
   }

   /**
    * Returns a <code>String</code> representation of this object
    * @return The <code>String</code> representation of this object
    */
   public String toString(){
     return "Packet Factory";
   }

}//end of PacketFactory
```

APPENDIX M – SAAM CONTROL.MAINGUI CLASS CODE

```
//23Feb2000[Henry]            - modified

package saam.control;

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.plaf.metal.MetalBorders.*;

import saam.util.*;

class MainGui extends JFrame {//implements Runnable old

    JToolBar toolbar;
    JMenuBar menubar;
    JMenuItem exit;
    JMenu fileMenu, protocolStackMenu, routingTableMenu,
      openChannelMenu, activePortMenu;
    JMenu slsTableMenu;  //Henry
    JMenu flowTableMenu;  //Henry
    Vector activeChannels = new Vector();
    Vector objectsToDisplay = new Vector();
    Vector tablesToDisplay = new Vector();
    Vector channelsToDisplay = new Vector();
    Vector portsToDisplay = new Vector();
    Vector slsTableToDisplay = new Vector();  //Henry
    Vector flowTableToDisplay = new Vector();  //Henry
      String[] columnNames = {"FlowRequest_Source",
"FlowRequest_Source",

      "ServiceType","FlowRequest_Throughput","FlowResponse_Result"};
      int[] columnWidths = {220,220,50,50,50};
      String flowTableTitle = "FlowRequest/FlowResponse Table";
    SoftTableGui flowTableGui;
    JPanel currentDisplay;
    ControlExecutive controlExec;
    String title;

  MainGui(ControlExecutive controlExec, String title){
    this.controlExec=controlExec;
    this.title=title;
    setTitle(title);
    createFileMenu();
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
    Dimension dim =
```

```
      Toolkit.getDefaultToolkit().getScreenSize();
    float screenFactor = 1.3f;
    setSize((int)(dim.width/(screenFactor)),
          (int)(dim.height/(screenFactor)));
    setLocation((int)(dim.width/2)-(int)(dim.width/(screenFactor)/2),
                (int)(dim.height/2)-
(int)(dim.height/(screenFactor)/2));
//        addImages();
    addRoots();
    updateProtocolStackObjects();
    setVisible(true);
    Thread mainGuiThread = new Thread(title);
    mainGuiThread.start();
  }
 // public void run(){
 //  }
  private void setCurrentDisplay(JPanel panel){
    if (currentDisplay!=null){
      currentDisplay.setVisible(false);
    }
    setTitle("Currently displaying: "+ panel.toString());
    currentDisplay = panel;
    currentDisplay.setBorder(BorderFactory.createEtchedBorder());
    setContentPane(new JScrollPane(currentDisplay));
    currentDisplay.setVisible(true);
    panel.validate();
    validate();
  }
  void createFileMenu(){
    menubar = new JMenuBar();
    fileMenu = new JMenu("File");
    exit = new JMenuItem("Exit");
    exit.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ae){
        System.exit(0);
      }
    });
    fileMenu.add(exit);
    protocolStackMenu = new JMenu("Protocol Stack");
    routingTableMenu = new JMenu("Routing Tables");
    openChannelMenu = new JMenu("Open Channels");
    activePortMenu = new JMenu("Active Ports");
    slsTableMenu = new JMenu("SLSTable");  //Henry
    flowTableMenu = new JMenu("Flow Tables");  //Henry
    menubar.add(fileMenu);

    menubar.add(protocolStackMenu);
    menubar.add(routingTableMenu);
    menubar.add(openChannelMenu);
    menubar.add(activePortMenu);
    menubar.add(slsTableMenu);  //Henry
    menubar.add(flowTableMenu);  //Henry
    setJMenuBar(menubar);

  }
  synchronized void updateDisplay(){
    updateRoutingTables();
```

```
    updateProtocolStackObjects();
    updateChannels();
    updateRouterSLSTable();  //Henry
  }

  void updateSLSTables(){  //Henry
    Vector titlesInGui = SLSTableGui.getTitles();
    for (int i=0;i<titlesInGui.size();i++){
      String thisTitle = (String)titlesInGui.get(i);
      if(!slsTableToDisplay.contains(thisTitle)){
        JMenuItem item = new JMenuItem(thisTitle);
        slsTableMenu.add(item);
        item.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent ae){
              setCurrentDisplay(SLSTableGui.getInstance(
                ae.getActionCommand()));
          }
        });
        slsTableToDisplay.add(thisTitle);
      }
    }
  }

  void updateRouterSLSTable(){  //Henry
    Vector titlesInGui = SLSTableGui.getTitles();
    for (int i=0;i<titlesInGui.size();i++){
      String thisTitle = (String)titlesInGui.get(i);
      if(!slsTableToDisplay.contains(thisTitle)){
        JMenuItem item = new JMenuItem(thisTitle);
        slsTableMenu.add(item);
        item.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent ae){
              setCurrentDisplay(SLSTableGui.getInstance(
                ae.getActionCommand()));
          }
        });
        slsTableToDisplay.add(thisTitle);
      }
    }
  }

  void updateFlowTables(Vector data){  //Henry
    flowTableGui = new SoftTableGui(flowTableTitle, columnNames,
columnWidths);
    flowTableGui.displayTableData(data);

    Vector titlesInGui = flowTableGui.getTitles();
    for (int i=0;i<titlesInGui.size();i++){
      String thisTitle = (String)titlesInGui.get(i);
      if(!flowTableToDisplay.contains(thisTitle)){
        JMenuItem item = new JMenuItem(thisTitle);
        flowTableMenu.add(item);
        item.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent ae){
              setCurrentDisplay(flowTableGui.getInstance(
                ae.getActionCommand()));
          }
```

```
        });
        flowTableToDisplay.add(thisTitle);
      }
    }
  }

  void updateRoutingTables(){
    Vector titlesInGui = TableGui.getTitles();
    for (int i=0;i<titlesInGui.size();i++){
      String thisTitle = (String)titlesInGui.get(i);
      if(!tablesToDisplay.contains(thisTitle)){
        JMenuItem item = new JMenuItem(thisTitle);
        routingTableMenu.add(item);
        item.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent ae){
              setCurrentDisplay(TableGui.getInstance(
                ae.getActionCommand()));
          }
        });
        tablesToDisplay.add(thisTitle);
      }
    }
  }
  void updateProtocolStackObjects(){

    Vector titlesInGui = SAAMRouterGui.getTitles();
    for (int i=0;i<titlesInGui.size();i++){
      String thisTitle = (String)titlesInGui.get(i);
      if(!objectsToDisplay.contains(thisTitle)){
        JMenuItem item = new JMenuItem(thisTitle);
        protocolStackMenu.add(item);
        item.addActionListener(new ActionListener(){
          public void actionPerformed(ActionEvent ae){
              setCurrentDisplay(SAAMRouterGui.getInstance(
                ae.getActionCommand()));
          }
        });
        objectsToDisplay.add(thisTitle);
      }
    }
  }
  void updateChannels(){
    ChannelTableGui gui = null;
    Enumeration activeChannels = controlExec.getActiveChannels();
    while(activeChannels.hasMoreElements()){
      Channel channel = (Channel)activeChannels.nextElement();
      int id = channel.getChannel_ID();
      Vector channelContents = channel.getChannel();
      String thisChannel = ""+id;
      gui = new ChannelTableGui(thisChannel,
        channel.getColumnHeaders(),
        channel.getColumnWidths());
      gui.fillTable(channelContents);
      if(id>controlExec.MAX_PORT){
        updateChannelDisplay(thisChannel);
      }else{
        updatePortDisplay(thisChannel);
```

```
    }
  }//while
}
private void updateChannelDisplay(String thisChannel){

    if(!channelsToDisplay.contains(thisChannel)){
      JMenuItem item = new JMenuItem(thisChannel);
      openChannelMenu.add(item);
      item.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent ae){
            setCurrentDisplay(ChannelTableGui.getInstance(
              ae.getActionCommand()));
        }
      });
      channelsToDisplay.add(thisChannel);
    }//if

}
private void updatePortDisplay(String thisChannel){

    if(!channelsToDisplay.contains(thisChannel)){
      JMenuItem item = new JMenuItem(thisChannel);
      activePortMenu.add(item);
      item.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent ae){
            setCurrentDisplay(ChannelTableGui.getInstance(
              ae.getActionCommand()));
        }
      });
      channelsToDisplay.add(thisChannel);
    }//if

}
private void addImages(){
  ImageCanvas imagePanel = new ImageCanvas(
                "D:\\tenchi.jpg",
                "Cary, you must FIGHT!");
  imagePanel.setBorder(
    BorderFactory.createTitledBorder("Tenchi!"));

  Container contentPane = getContentPane();
  contentPane.setLayout(new FlowLayout());
  contentPane.add(imagePanel);

  imagePanel = new ImageCanvas(
                "D:\\kiyone.jpg",
                "Cary, you must FIGHT!");
  imagePanel.setBorder(
    BorderFactory.createTitledBorder("Kiyone!"));
  contentPane.add(imagePanel);

  imagePanel = new ImageCanvas(
                "D:\\sasami.jpg",
                "Cary, you must FIGHT!");
  imagePanel.setBorder(
    BorderFactory.createTitledBorder("Sasami!"));
  contentPane.add(imagePanel);
```

```
    imagePanel = new ImageCanvas(
                 "D:\\aeka.jpg",
                 "Cary, you must FIGHT!");
    imagePanel.setBorder(
      BorderFactory.createTitledBorder("Aeka!"));
    contentPane.add(imagePanel);

    imagePanel = new ImageCanvas(
                 "D:\\washu.jpg",
                 "Cary, you must FIGHT!");
    imagePanel.setBorder(
      BorderFactory.createTitledBorder("Washu!"));
    contentPane.add(imagePanel);

  }
  private void addRoots(){
    ImageCanvas imagePanel = new ImageCanvas(
                 "images"+
                 File.separatorChar+"workhard.jpg",
                 "Cary, you must FIGHT!");
    imagePanel.setBorder(
      BorderFactory.createTitledBorder(
        new Flush3DBorder(),
        "Java's a piece of cake, it just takes a little time!",
        TitledBorder.CENTER,
        TitledBorder.BELOW_TOP));

    Container contentPane = getContentPane();
    contentPane.setLayout(new FlowLayout());
    contentPane.add(imagePanel);
  }
  class ImageCanvas extends JPanel {
    ImageIcon icon;

    public ImageCanvas(String imageName, String description) {
      icon = new ImageIcon(imageName, description);
    }
    public void paintComponent(Graphics g) {
      Insets insets = getInsets();
      super.paintComponent(g);
      icon.paintIcon(this, g, insets.left, insets.top);
    }
    public Dimension getPreferredSize() {
      Insets insets = getInsets();
      return new Dimension(
        icon.getIconWidth() + insets.left + insets.right,
        icon.getIconHeight() + insets.top + insets.bottom);
    }
  }

}//end of MainGui class
```

APPENDIX N – SAAM UTIL.FILEIO CLASS CODE

```
// 10Jan2000[Henry]           - Created

package saam.util;

import java.util.*;
import java.io.*;

/**
 * The <em>FileIO</em> is an object for file read/write operations
 */

public class FileIO {

  private BufferedReader bufReader;
  private PrintWriter bufWriter;
  private File file;
  private StringTokenizer st;

  /**
   * Constructs a FileIO object without any arguments.
   */
  public FileIO() {
  }

  /**
   * To open a tile for reading
   * @param   filename
   */
  public void openToRead(String filename) {

    //file = new File("..\\saam\\"+filename);//for Kawa project
    file = new File(filename);//for Kawa project
    try {
      bufReader = new BufferedReader(new
FileReader(file.getAbsoluteFile()));
    }
    catch (FileNotFoundException fnf) {
      System.err.println("FileNotFoundException - not Kawa project");
      System.err.println(file.getAbsoluteFile());
      filename = "..\\..\\"+filename;//for Jbuilder project
      openToRead(filename);
    }
    catch (IOException ioe) {
      System.err.println("IOException");
    }
  }

  /**
   * To open a file for writing
   * @param   filename
   */
  public void openToWrite(String filename) {
```

```
    //file = new File("..\\saam\\"+filename);//for Kawa project
    file = new File(filename);//for Kawa project
    try {
      bufWriter = new PrintWriter(new
FileWriter(file.getAbsoluteFile()));
    }
    catch (FileNotFoundException fnf) {
      System.err.println("FileNotFoundException - not Kawa project");
      System.err.println(file.getAbsoluteFile());
      filename = "..\\..\\"+filename;//for Jbuilder project
      openToWrite(filename);
    }
    catch (IOException ioe) {
      System.err.println("IOException");
    }
  }

  /**
   * To read one line of data at a time from the file which
   * has been opened for reading
   * @return  The data string
   */
  public String readLine() {
    String input = null;
    try {
      input = bufReader.readLine();
    }
    catch (IOException ioe) {
      System.err.println("IOException");
    }
    return input;
  }

  /**
   * To write an object to the file which has been opened
   * for writing
   * @param   obj
   */
  public void write(Object obj) {
    bufWriter.print(obj);
    bufWriter.flush();
  }

  /**
   * To close the file which has been opened
   */
  public void close() {
    try {
      if (bufReader != null) {
        bufReader.close();
      }
      else {
        bufWriter.close();
      }
    }
    catch (IOException ioe) {
      System.err.println("IOException");
```

```
    }
  }
}//end of FileIO class
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX O – SAAM.DEMO.DEMO_1SERVER_1ROUTER CLASS CODE

```
//10Mar99[Henry]  - Created

package saam.demo;

import saam.*;
import saam.control.*;
import saam.message.*;
import saam.residentagent.*;
import saam.router.*;
import saam.net.*;
import saam.util.*;
import java.net.*;
import java.io.*;
import java.util.Vector;
import java.util.Enumeration;

import saam.server.*;
import saam.server.diffserv.*;

public class Demo_1Server_1Router{
   private PacketFactory packet = new PacketFactory();

   private InetAddress
destMain,destBackUp,destA,destB,destC,destD,destE; //IPv4s of ROUTERs
to stand-up
   private int destEmulationPort=9002;   //SAAM UDP emulation port
(IPv4 world)
   private DemoGui gui = new DemoGui("Demo_1Server_1Router");

   Configuration cfMain = null;  //FOR MAIN SERVER
   Configuration cfBackUp = null; //For BackUp Server

   //Different values may be sent to each server and router
   private int timeScaleForMain = 250;
   private int timeScaleForBackUp = 250;
   private int timeScaleForRouter_A = 250;
  // private int timeScaleForRouter_B = 300;
  // private int timeScaleForRouter_C = 400;
  // private int timeScaleForRouter_D = 400;
  // private int timeScaleForRouter_E = 400;

   private static final byte MAIN_SERVER_TYPE_ID = 0;
   private static final byte BACK_UP_SERVER_TYPE_ID = 1;

   private static final int MAIN_SERVER_FLOW_ID = 1;
   private static final int BACK_UP_SERVER_FLOW_ID = 3;

   private static final byte METRIC_TYPE = 0;
   //Metric Type 0-> For Symmetric ( first arriving best), 1-> For
Hopcount

   private static final int MAIN_REFRESH_CYCLE_TIME = 300;// In msec.
```

```
   private static final int BACK_UP_REFRESH_CYCLE_TIME = 300;// In
msec.

   private static final int MAIN_GLOBALTIME_TO_WAIT = 200;// In msec.
   private static final int BACK_UP_GLOBALTIME_TO_WAIT = 200;// In
msec.

   // coreAgents are those resident agents which all emulated players
   // must receive to stand-up
   private String[] coreAgents =
     {"saam.residentagent.router.Scheduler",
      "saam.residentagent.router.ARPCache",
      "saam.residentagent.router.FlowRoutingTable"};

   private String redwood = "131.120.8.153";
   private String pine = "131.120.8.137";
   private String cherry = "131.120.8.143";
   private String oak = "131.120.8.136";
   private String sumatra = "131.120.8.134";
   private String dogwood = "131.120.8.132";
   private String maple = "131.120.8.142";

   private String serV6 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.1";
   private String serNextHopV6 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.2";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.2";
//For SP238
   private String serNextHopV4 = pine;
   private String primaryServer = maple;
/*For SP525
   private String serNextHopV4 = dogwood;
   private String primaryServer = sumatra;
*/
   private String routerAV6 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.2";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.2";
   private String routerANextHopV6 =
"99.99.99.99.0.0.0.0.0.0.0.0.0.0.1";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.1";
//For SP238
   private String routerANextHopV4 = maple;
   private String routerAV4 = pine;
/*For SP525
   private String routerANextHopV4 = sumatra;
   private String routerAV4 = dogwood;
*/

   public static void main(String args[]){
       Demo_1Server_1Router test = new  Demo_1Server_1Router();
       System.exit(0);
   }

   public  Demo_1Server_1Router(){

    try{
     gui.setTextField("My IP: "+
InetAddress.getLocalHost().getHostAddress());
    }catch(UnknownHostException uhe){
```

```
     gui.sendText(uhe.toString());
    }

    try{
       destMain= InetAddress.getByName(primaryServer); //server
       destBackUp= InetAddress.getByName("131.120.8.132");//backup
       destA= InetAddress.getByName(routerAV4); //Router A
     // destB= InetAddress.getByName("131.120.8.139"); //Router B
     //  destC= InetAddress.getByName("131.120.9.76"); //Router C
     //  destD= InetAddress.getByName("131.120.9.76"); //Router D
     //  destE= InetAddress.getByName("127.0.0.1"); //Router E
    }catch(UnknownHostException uhe){
      gui.sendText(uhe.toString());
    }


//FIRST STAND UP SERVER!!!!!!
    //Initilizing interfaces on MAIN Server
    Vector serverInterface = new Vector();
    Vector serverEmTable = new Vector();
    Vector serverArpCache = new Vector();
    IPv6Address serIntAd = null;

    byte  serverMac = 0;
    byte  serverNextMac = 1;
    try{
    serIntAd = new IPv6Address(
       IPv6Address.getByName(serV6).getAddress());
    IPv6Address serNextHop = new IPv6Address(
       IPv6Address.getByName(serNextHopV6).getAddress());
    InetAddress serNextV4 = InetAddress.getByName(serNextHopV4);

    //for demohello message
    serverInterface.add( new InterfaceID( serIntAd,serverMac));
    //for EmulationTableEntry message
    serverEmTable.add(new EmulationTableEntry(serNextHop,serNextV4));
    //for ARPCache
    serverArpCache.add( new  ARPCacheEntry(serNextHop,serverNextMac));

    }
    catch(UnknownHostException uhe){
    gui.sendText(uhe.toString());
    }

    cfMain = new Configuration(MAIN_SERVER_TYPE_ID,
MAIN_SERVER_FLOW_ID,
               METRIC_TYPE, MAIN_REFRESH_CYCLE_TIME*timeScaleForMain,
              MAIN_GLOBALTIME_TO_WAIT*timeScaleForRouter_A );
              //actualy any router not specificaly A

//NOW STAND-UP THE ROUTERS!!!!!!!!!
    //ROUTER A
    Vector routerAInterfaces = new Vector();
    Vector routerAEmTable = new Vector();
    Vector routerAArpCache = new Vector();
    //interface-1
    byte routerAMacs_1 = 1;
```

```
    byte routerANextMac_1 = 0;
    IPv6Address routerAInt_1 = null;
    try{
      routerAInt_1 = new IPv6Address(IPv6Address.getByName(
                              routerAV6).getAddress());
      IPv6Address routerANextHop_1 = new IPv6Address(
                          IPv6Address.getByName(serV6).getAddress());
      InetAddress routerANextV4_1 =
InetAddress.getByName(primaryServer);

      routerAInterfaces.add( new
InterfaceID(routerAInt_1,routerAMacs_1));
      routerAEmTable.add(new EmulationTableEntry(
                          routerANextHop_1,routerANextV4_1));
      routerAArpCache.add( new  ARPCacheEntry(
                          routerANextHop_1,routerANextMac_1));
    }
    catch(UnknownHostException uhe){
     gui.sendText(uhe.toString());
    }

    // start Primary Server
    InitServer(serverInterface, serverEmTable, serverArpCache,
                                destMain, cfMain);
    // start Router A
    InitRouter( routerAInterfaces, routerAEmTable, routerAArpCache,
                         destA, timeScaleForRouter_A );

    try{
      Thread.sleep(10000);
    }catch(InterruptedException ie){
      gui.sendText("problem afetr initrouter thread sleep");
    }

  }//end DemoStation() constructor


   public void InitRouter(Vector routerInterfaces, Vector
routerEmTable,
      Vector routerArpCache, InetAddress dest, int tsForRouter){

      //add router InterfaceIDs  -- may have to use DemoHello
      //messages instead
      DemoHello helloMessage = new DemoHello(routerInterfaces);
      packet.append(helloMessage);

      try{
        //now append some ResidentAgents...
        //first the agents that are necessary for the
        //protocol stack
        for(int i=0;i<coreAgents.length;i++){
         packet.append(coreAgents[i]);
        }

        //then any additional agents for the specific host
        packet.append("saam.residentagent.router.SLSTable");
```

```
      }catch(IOException ioe){
         gui.sendText(ioe.toString());
         gui.sendText(" problem in initrouter coreagents for block ");
      }

      packet.append(new TimeScale(tsForRouter));
      //add entries to the EmulationTable
      Enumeration e1 = routerEmTable.elements();
      while(e1.hasMoreElements()){
         packet.append( (EmulationTableEntry)( e1.nextElement()));

      } //end of while

      //add entries to the ARPCache
       Enumeration e2 = routerArpCache.elements();
      while(e2.hasMoreElements()){
         packet.append((ARPCacheEntry) e2.nextElement());

      } //end of while


      // by passing 250 from header of method, i can set it
      // diffrent for each router

      //now send the packet
      byte[] packetArray = packet.getBytes();
      //Note:  getBytes also sets packet object up to be reused for a
new message
      gui.sendText("#of messages: "+packetArray[8]);//peeks inside
packet
      try{
         Socket socket = new Socket(dest,destEmulationPort);
         socket.setTcpNoDelay(true);


         gui.sendText("destRouter = "+dest+" destEmuPort= "+
destEmulationPort);
         OutputStream os = socket.getOutputStream();

         os.write(packetArray);
        // os.flush();
         os.close();
         socket.close();
      }catch(Exception e){
        gui.sendText(e.toString());
        gui.sendText(" problem in initrouter socket try block ");
      }
      gui.sendText("Packet sent to "+dest.getHostAddress());
      gui.sendText("Length: "+packetArray.length);

     try{
       Thread.sleep(packetArray.length);
     }
     catch(InterruptedException ie){
     }

   }//end InitRouter()
```

```
   public void InitServer( Vector serverInterface, Vector
serverEmTable,
     Vector serverArpCache, InetAddress destS, Configuration cf){


      DemoHello helloMessage = new DemoHello(serverInterface);
      packet.append(helloMessage);

      try{
        //now append some ResidentAgents...
        //first the agents that are necessary for the
        //protocol stack
        for(int i=0;i<coreAgents.length;i++){
         packet.append(coreAgents[i]);
        }
        //then any additional agents for the specific host
        packet.append("saam.residentagent.server.ServerAgentSymetric");
       //
packet.append("saam.residentagent.server.ServerAgentHopCount");


      }catch(IOException ioe){
         gui.sendText(ioe.toString());
      }

      //add entries to the Server's EmulationTable
      Enumeration e1 = serverEmTable.elements();
      while(e1.hasMoreElements()){
         packet.append( (EmulationTableEntry)( e1.nextElement()));

      } //end of while

      //add entries to the Server's ARPCache

      Enumeration e2 = serverArpCache.elements();
      while(e2.hasMoreElements()){
         packet.append( (ARPCacheEntry)( e2.nextElement()));

      } //end of while

      //TO SEND CONFIGURATION INFORMATION

      packet.append( cf );

      //now send the packet
      byte[] packetArray = packet.getBytes();
      //Note: getBytes also sets packet object up to be reused
      //for a new message
      gui.sendText("#of messages send to server :
"+packetArray[8]);//peeks inside packet
      try{
            Socket socket = new Socket(destS,destEmulationPort);
            socket.setTcpNoDelay(true);
        gui.sendText("destServer  =  "+destS+" destEmuPort= "+
destEmulationPort);
        OutputStream os = socket.getOutputStream();
```

```
            os.write(packetArray);
        //   os.flush();
            os.close();
            socket.close();
        }catch(Exception e){
          gui.sendText(e.toString());
          gui.sendText(" problem in initserver socket try block ");
        }

        gui.sendText("Packet sent to "+destS.getHostAddress());
        gui.sendText("Length: "+packetArray.length);

    }//end InitServer()

}//end class Demo_1Server_1Router
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX P – SAAM.DEMO.SENDFLOWAGENT CLASS CODE

```
//10Mar99[Henry]  - Created

package saam.demo;

import saam.*;
import saam.control.*;
import saam.message.*;
import saam.residentagent.*;
import saam.router.*;
import saam.net.*;
import saam.util.*;
import java.net.*;
import java.io.*;
import java.util.Vector;
import java.util.Enumeration;

import saam.server.*;
import saam.server.diffserv.*;

/**
 * A class that may be used to send resident agent(s) to the routers
 */
public class SendFlowAgent{
   private PacketFactory packet = new PacketFactory();

   private InetAddress
destMain,destBackUp,destA,destB,destC,destD,destE; //IPv4s of ROUTERs
to stand-up
   private int destEmulationPort=9002;   //SAAM UDP emulation port
(IPv4 world)
   private DemoGui gui = new DemoGui("SendFlowAgent");

   Configuration cfMain = null;  //FOR MAIN SERVER
   Configuration cfBackUp = null; //For BackUp Server

   //Different values may be sent to each server and router
   private int timeScaleForMain = 25;
   private int timeScaleForBackUp = 25;
   private int timeScaleForRouter_A = 25;
  // private int timeScaleForRouter_B = 300;
  // private int timeScaleForRouter_C = 400;
  // private int timeScaleForRouter_D = 400;
  // private int timeScaleForRouter_E = 400;

   private static final byte MAIN_SERVER_TYPE_ID = 0;
   private static final byte BACK_UP_SERVER_TYPE_ID = 1;

   private static final int MAIN_SERVER_FLOW_ID = 1;
   private static final int BACK_UP_SERVER_FLOW_ID = 3;

   private static final byte METRIC_TYPE = 0;
   //Metric Type 0-> For Symmetric ( first arriving best), 1-> For
Hopcount
```

```
   private static final int MAIN_REFRESH_CYCLE_TIME = 2000;// In msec.
   private static final int BACK_UP_REFRESH_CYCLE_TIME = 2000;// In
msec.

   private static final int MAIN_GLOBALTIME_TO_WAIT = 200;// In msec.
   private static final int BACK_UP_GLOBALTIME_TO_WAIT = 200;// In
msec.

   // coreAgents are those resident agents which all emulated
   // players must receive to stand-up
   private String[] coreAgents =
     {"saam.residentagent.router.Scheduler",
      "saam.residentagent.router.ARPCache",
      "saam.residentagent.router.FlowRoutingTable"};

   private String redwood = "131.120.8.153";
   private String pine = "131.120.8.137";
   private String cherry = "131.120.8.143";
   private String oak = "131.120.8.136";
   private String sumatra = "131.120.8.134";
   private String dogwood = "131.120.8.132";
   private String maple = "131.120.8.142";

   private String serV6 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
   private String serNextHopV6 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.2";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.2";
//For SP238
   private String serNextHopV4 = pine;
   private String primaryServer = maple;
/*For SP525
   private String serNextHopV4 = dogwood;
   private String primaryServer = sumatra;
*/

   private String routerAV6 = "99.99.99.99.0.0.0.0.0.0.0.0.0.0.2";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.0.2";
   private String routerANextHopV6 =
"99.99.99.99.0.0.0.0.0.0.0.0.0.0.1";
           //"99.99.99.0.0.0.0.0.0.0.0.0.0.0.1";
//For SP238
   private String routerANextHopV4 = maple;
   private String routerAV4 = pine;
/*For SP525
   private String routerANextHopV4 = sumatra;
   private String routerAV4 = dogwood;
*/
   public static void main(String args[]){
       SendFlowAgent test = new  SendFlowAgent();
       System.exit(0);
   }

  public  SendFlowAgent(){

    try{
```

```
     gui.setTextField("My IP: "+
InetAddress.getLocalHost().getHostAddress());
    }catch(UnknownHostException uhe){
     gui.sendText(uhe.toString());
    }

    try{
       destMain= InetAddress.getByName(primaryServer); //server
       destBackUp= InetAddress.getByName("131.120.8.132");//backup
       destA= InetAddress.getByName(routerAV4); //Router A
     // destB= InetAddress.getByName("131.120.8.139"); //Router B
     //  destC= InetAddress.getByName("131.120.9.76"); //Router C
     //  destD= InetAddress.getByName("131.120.9.76"); //Router D
     //  destE= InetAddress.getByName("127.0.0.1"); //Router E
    }catch(UnknownHostException uhe){
      gui.sendText(uhe.toString());
    }

    //SendTestMessages to Server
    sendPacket(destMain, "saam.residentagent.router.OneWayDSFlow");

    //SendTestMessages to Router
    sendPacket(destA, "saam.residentagent.router.OneWayDSFlow");

  }//end sendFlowAgent() constructor


  /**
   * Gets the IPv6Address equivalent of the string given in the
   * parameter
   * @param   node  The IPv6Address in the form of a string
   * @return  IPv6Address equivalent
   */
   private IPv6Address getV6Address(String node) {
      IPv6Address address = null;
      try{
         address = new IPv6Address(
IPv6Address.getByName(node).getAddress());
      }
      catch(UnknownHostException uhe){
         gui.sendText(uhe.toString());
      }
      return address;
   }

  /**
   * Sends the resident agent class file to the destination
   * specified.
   * @param   destS  The IPv4Address of the destination
   * @param   residentAgentName
   */
      private void sendPacket(InetAddress destS, String
residentAgentName) {

            //Test resident agent
            try{
                  packet.append(new TestMessage(routerAV4));
```

```
                    packet.append(residentAgentName);
    }catch(IOException ioe){
       gui.sendText(ioe.toString());
    }

    //now send the packet
    byte[] packetArray = packet.getBytes();

    //Note:  getBytes also sets packet object up to be reused for a new
message
    gui.sendText("#of messages sent: "+packetArray[8]);//peeks inside
packet
    try{
       Socket socket = new Socket(destS,destEmulationPort);
       socket.setTcpNoDelay(true);
       gui.sendText("dest =  "+destS+" destEmuPort = "+
destEmulationPort);
       OutputStream os = socket.getOutputStream();

       os.write(packetArray);
    //   os.flush();
       os.close();
       socket.close();
    }catch(Exception e){
      gui.sendText(e.toString());
      gui.sendText(" problem in initserver socket try block ");
    }
    gui.sendText("Packet sent to "+destS.getHostAddress());.
    gui.sendText("Length: "+packetArray.length);

   try{
     Thread.sleep(packetArray.length);
   }
   catch(InterruptedException ie){
   }

  }//sendPacket

}//end class SendFlowAgent
```

APPENDIX Q – SAAM.DEMO.QOSDEMO PACKAGE CODE

```
// 13Feb2000, Henry    - Created

package saam.demo.QoSDemo;

/**
 * The <em>QoSDemo</em> class is the main class used to test
 * and verify the QoS Management classes and their functions.
 */
public class QoSDemo {

  public static void main(String args[]){
    FourNodes myTopology = new FourNodes();
    myTopology.start();
  }//main

}//end QoSDemo
```

```
// 1Feb2000, Henry    - Modified
// 14Dec1999, Henry    - Created

package saam.demo.QoSDemo;

import java.net.UnknownHostException;
import java.util.*;

import saam.message.*;
import saam.control.*;
import saam.server.*;
import saam.net.*;
import saam.util.*;
import saam.server.diffserv.*;

/**
 * The <em>QoSDemo</em> class is used to test and verify the SLS
 * classes.
 */
public class FourNodes extends Thread{

  SLS sls;
  SLSDbase SLS_dbase;
  FlowResponse response;
  FlowRequest  request;
  Object[] IP = null;
  Random randomGen;
  int supporting_path = 0;
  int numberOfNodes = 7;
  int numberOfInterfaces = 0;
  int numberOfLinks = 4;
  int[] node_ids;// = new int[numberOfNodes];
  IPv6Address[] address;
  boolean localTest = false;

  ControlExecutive ce;
  Server  server;
  ClassObjectStructure PIB;
  private SAAMRouterGui gui;
  private DemoGui dgui;
      private NodeThread node;


  /**
   * Construct a four node topology for local testing
   */
  public FourNodes(){
    //randomGen = new Random();
    dgui = new DemoGui("IntServ & DiffServ DemoStation");
    gui = new SAAMRouterGui("FourNodes");
    ce = new ControlExecutive();
    PIB = new ClassObjectStructure();
    SLS_dbase = new SLSDbase();
    PIB.deleteAllData();
    server = new Server(gui, ce, PIB, SLS_dbase);
    localTest = true;
  }
```

```
  /**
   * Construct a four node topology for integrated testing
   */
  public FourNodes(Server server){
    dgui = new DemoGui("IntServ & DiffServ DemoStation");
    gui = new SAAMRouterGui("FourNodes");
    this.server = server;
  }

      /**
       * Starts running the demo test
       */
  public void run(){
      RequesterThread requester;
    dgui.setTextField("Initial resources ...");
    setup();
    if (localTest) {
      for (int i=0; i<2; i++) {  //four IntServ request
        requester = new RequesterThread(dgui, server,
                (IPv6Address)IP[0], (IPv6Address)IP[4], 2000, 1000);
        requester.start();
      }
      for (int i=0; i<2; i++) {  //four IntServ request
        requester = new RequesterThread(dgui, server,
                (IPv6Address)IP[0], (IPv6Address)IP[4],
                  Server.DS_SERVICELEVEL, 1000, 1000);
        requester.start();
      }
      //requester = new RequesterThread(dgui, server, address, 1000,
1000);
      //requester.start();

      requester = new RequesterThread(dgui, server,
              (IPv6Address)IP[0], (IPv6Address)IP[5], 1050, 1000);
      requester.start();

      //unreacheable path
      requester = new RequesterThread(dgui, server,
              (IPv6Address)IP[0], (IPv6Address)IP[7], 2050, 1000);
      requester.start();
      requester = new RequesterThread(dgui, server,
              (IPv6Address)IP[0], (IPv6Address)IP[7],
                  Server.DS_SERVICELEVEL, 1000, 1000);
      requester.start();
    }
    requester = new RequesterThread(dgui, server,
                        (IPv6Address)IP[9], (IPv6Address)IP[10],
                                Server.IS_SERVICELEVEL, 100, 100);
    requester.start();

    gui.setTextField("done");
    try{
      Thread.sleep(100000);
    }
    catch(InterruptedException ie){
      gui.sendText(ie.toString());
```

```
    }

  }//end run()


      /**
       * Setup the nodes and its interfaces
       */
  private void setup(){
    gui.sendText("Setting ...");
    //int node_id;// = 0;     //is node 1
    //int interfaceID = 0;
    addIP();
    //Server
    node = new NodeThread(server, Array.getSubArray(IP,10,11));
    //RouterA
    node = new NodeThread(server, Array.getSubArray(IP,0,3));
    //node.start();
    //RouterB
            node = new NodeThread(server, Array.getSubArray(IP,3,5));
            //node.start();
    //RouterC
            node = new NodeThread(server, Array.getSubArray(IP,5,7));
            //node.start();
            //node = new NodeThread(server, Array.getSubArray(IP,7,9));
            //node.start();
            //node = new NodeThread(server,
Array.getSubArray(IP,9,10));
            address = new IPv6Address[numberOfInterfaces];
            //address = (IPv6Address[])IP;
  }


  /**
   * Construct IPs that will reside on routers
   */
  private void addIP(){
    try{
                  numberOfInterfaces++;
      IP = Array.concat(IP,
        IPv6Address.getByName(//Node 1
//           "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1"));
          "99.99.99.1.0.0.0.0.0.0.0.0.0.0.0.1"));
                  numberOfInterfaces++;
      IP = Array.concat(IP,
        IPv6Address.getByName(//Node 1
//           "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2"));
          "99.99.99.3.0.0.0.0.0.0.0.0.0.0.0.2"));
                  numberOfInterfaces++;
      IP = Array.concat(IP,
        IPv6Address.getByName(//Node 1
//           "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.3"));
          "99.99.99.0.0.0.0.0.0.0.0.0.0.0.0.2"));
                  numberOfInterfaces++;
      IP = Array.concat(IP,
        IPv6Address.getByName(//Node 2
//           "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1"));
```

```
            "99.99.99.2.0.0.0.0.0.0.0.0.0.0.0.1"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 2
//              "99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2"));
            "99.99.99.1.0.0.0.0.0.0.0.0.0.0.0.2"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 3
//              "99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1"));
            "99.99.99.3.0.0.0.0.0.0.0.0.0.0.0.1"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 3
//              "99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2"));
            "99.99.99.2.0.0.0.0.0.0.0.0.0.0.0.2"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 4
//              "99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1"));
            "99.99.99.4.0.0.0.0.0.0.0.0.0.0.0.1"));
                  /*numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 4
            "99.99.99.3.0.0.0.0.0.0.0.0.0.0.0.2"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 4
            "99.99.99.6.0.0.0.0.0.0.0.0.0.0.0.3"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 5
            "99.99.99.4.0.0.0.0.0.0.0.0.0.0.0.2"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 5
            "99.99.99.5.0.0.0.0.0.0.0.0.0.0.0.1"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 6
            "99.99.99.6.0.0.0.0.0.0.0.0.0.0.0.1"));
                  numberOfInterfaces++;*/
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 7
//              "99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1"));
            "99.99.99.7.0.0.0.0.0.0.0.0.0.0.0.1"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Node 7 to Server
//              "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.2"));
            "99.99.99.0.0.0.0.0.0.0.0.0.0.0.0.2"));
                  numberOfInterfaces++;
        IP = Array.concat(IP,
          IPv6Address.getByName(//Server
//              "99.99.99.99.0.0.0.0.0.0.0.0.0.0.0.1"));
            "99.99.99.0.0.0.0.0.0.0.0.0.0.0.0.1"));
    }
```

```
    catch(UnknownHostException uhe){
      gui.sendText(uhe.toString());
      System.err.println(uhe.toString());
    }
  }


  /**
   * Print the remainingThroughput of all interfaces of the service
   * level specified in the parameter.
   * @param   service_level  The service level interested
   */
  private void displayInterfaceStatus(byte service_level){
    for (int i=0; i<numberOfInterfaces; i++) {
      address[i] = (IPv6Address)IP[i];
      dgui.sendText("Interface: " +address[i]
        + "'s remainingThroughput = "+
            PIB.getRemainingThroughput(address[i], service_level));
    }
  }


  /**
   * Print the remainingThroughput of all interfaces of all service
   * level.
   */
  private void displayAllInterfaceStatus(){
    for (int i=0; i<numberOfInterfaces; i++) {
      address[i] = (IPv6Address)IP[i];
      for (byte service_level = 0;
            service_level<Server.NUMBEROFSERVICELEVELS;
service_level++){
              dgui.sendText("Interface: " +address[i]
            + "'s remainingThroughput = "+
                PIB.getRemainingThroughput(address[i],
service_level));
                }
    }
  }


}//end FourNodes
```

```
//-----------------------------------------------------------------------
// Filename    : NodeThread.java
// Date        : January 27, 2000
//-----------------------------------------------------------------------

package saam.demo.QoSDemo;

import java.net.UnknownHostException;
import java.util.*;
import saam.message.*;
import saam.control.*;
import saam.server.*;
import saam.net.*;
import saam.util.*;
import saam.server.diffserv.*;

/**
 * NodeThread class used to simulate a router which will establish
 * contact with the server using a Hello message.
 */
public class NodeThread extends Thread {

 /**
  * counter is an static integer and used for assigning thread numbers
  */
  private static byte counter = 0;

 /**
  * threadNumber is an integer and used for thread numbers
  */
  private byte threadNumber = 0;

  private Server server;
  private Random randomGen = new Random();
  private Object[] address;
  private Vector interfaces = new Vector();
  private Hello hello;
  private SAAMRouterGui gui;

 /**
  * Constructor of this class.
  */
  //Used only by server
  public NodeThread(Server server, IPv6Address address) {
    threadNumber = ++counter;
    //gui = new SAAMRouterGui("NodeThread"+threadNumber);
    this.server = server;
    this.address = new Object[1];
    this.address[0] = (Object)address;
    InterfaceID interfaceId =
        new InterfaceID(address,  10000, threadNumber);
    interfaces.add(interfaceId);
            hello = new Hello(interfaces);
    //simulate server received Hello message from the router
            server.processHello(hello);
  }//end NodeThread()
```

```
  public NodeThread(Server server, Object[] address) {
    threadNumber = ++counter;
    //gui = new SAAMRouterGui("NodeThread"+threadNumber);
    this.server = server;
    this.address = address;
    for (int i=0; i<address.length; i++) {
      InterfaceID interfaceId =
                  new InterfaceID((IPv6Address)address[i],  10000,
threadNumber);
      interfaces.add(interfaceId);
    }
            hello = new Hello(interfaces);
    //simulate server received Hello message from the router
            server.processHello(hello);
  }//end NodeThread()


  /**
   * Used to simulate a flow request.
   */
  public void run() {
      try{
            Thread.sleep(1000);
          RequesterThread requester =
                       new RequesterThread(server, address, 1000,
100);
          requester.start();
    }catch(InterruptedException ie){}
  }

 /**
  * Returns the string representation of this class
  * @return string
  */
  public String toString(){
     return ("\nThread Number          : " + threadNumber +"");
  }//end toString()

}//end NodeThread class

//end file NodeThread.java
```

```
//-----------------------------------------------------------------------
// Filename    : RequesterThread.java
// Date        : January 27, 2000
//-----------------------------------------------------------------------

package saam.demo.QoSDemo;

import java.net.UnknownHostException;
import java.util.*;
import saam.message.*;
import saam.control.*;
import saam.server.*;
import saam.net.*;
import saam.util.*;
import saam.server.diffserv.*;

/**
 * RequesterThread class simulates an application requiring a service
 * flow. The sequence of requesting a service flow and responding to
 * the response from the server is started by instantiating this thread
 * and calling its start() method.
 */
public class RequesterThread extends Thread {

 /**
  * counter is an static integer and used for assigning thread numbers
  */
  private static int counter = 0;

 /**
  * threadNumber is an integer and used for thread numbers
  */
  private int threadNumber = 0;

 /**
  * processTime is an integer and used for required process time of
  * the thread
  */
  private int processTime;

  private Random randomGen = new Random();
  private Server server;
  private Object[] address;
  private IPv6Address source;
  private IPv6Address dest;
  private int throughput = 0;
  private   byte service_level = Server.IS_SERVICELEVEL;
  private boolean dynamic = false;
      private FlowRequest request;
      private FlowResponse response;
      private DemoGui gui;

 /**
  * Constructor of this class.
  */
  public RequesterThread(DemoGui gui,
      Server server, Object[] address,
```

```
            int throughput, int maxProcessTime) {
    this(server, address, throughput, maxProcessTime);
  this.gui = gui;
}

public RequesterThread(
    Server server, Object[] address,
            int throughput, int maxProcessTime) {

  threadNumber = ++counter;
  this.server = server;
  this.address = address;
  this.throughput = throughput;
  this.processTime = randomGen.nextInt(maxProcessTime);
  dynamic = true;

}//end RequesterThread()

public RequesterThread(DemoGui gui,
    Server server, IPv6Address source, IPv6Address dest,
            int throughput, int maxProcessTime) {

  threadNumber = ++counter;
  this.gui = gui;
  this.server = server;
  this.source = source;
  this.dest = dest;
  this.throughput = throughput;
  this.processTime = randomGen.nextInt(maxProcessTime);
  dynamic = false;

}//end RequesterThread()

public RequesterThread(DemoGui gui,
    Server server, IPv6Address source, IPv6Address dest,
          byte service_level, int throughput, int maxProcessTime) {

  threadNumber = ++counter;
  this.gui = gui;
  this.server = server;
  this.source = source;
  this.dest = dest;
  this.throughput = throughput;
  this.service_level = service_level;
  this.processTime = maxProcessTime;
    //randomGen.nextInt(maxProcessTime);
  dynamic = false;

}//end RequesterThread()


/**
 * Used to start the test sequence of sendng flow request and
 * then processing the flow response result.
 */
public void run() {
    if (service_level == Server.DS_SERVICELEVEL) {
```

```
            //make a DiffServ request
            gui.sendText("Requester: "+threadNumber+" is requesting
DiffServ....");
            request = new FlowRequest(source, dest,
                  System.currentTimeMillis(), threadNumber, 1, 1,
throughput);
            response = server.DS_Admission(request);
      }
      else {
            //make a IntServ request
            gui.sendText("Requester: "+threadNumber+" is requesting
IntServ....");
            if (dynamic) {
                  request = new FlowRequest(

      (IPv6Address)address[randomGen.nextInt(address.length)],

      (IPv6Address)address[randomGen.nextInt(address.length)],
                  Server.IS_SERVICELEVEL, System.currentTimeMillis(),
              1, 1, throughput);
            }
            else {
                  request = new FlowRequest(source, dest,
                        Server.IS_SERVICELEVEL,
System.currentTimeMillis(),
              1, 1, throughput);
            }
            response = server.IS_Admission(request);
      }
      byte result = response.getResult();
      if (result == FlowResponse.IS_ACCEPTED) {
            try {
                  gui.sendText("Requester: "+threadNumber+" is going
asleep for:"
                                    +processTime+" ms of simulated
traffic time.");
                  sleep(processTime);
                  gui.sendText("Requester: "+threadNumber+" is sending
flow termination.");
                  //simulate server receiving FlowTermination message
from router
        server.receiveFlowTermination(response.getFlowId());
            }
            catch (InterruptedException ie) {
                  //do nothing
                  gui.sendText(ie.toString());
            }
      }
      else if (result == FlowResponse.DS_ACCEPTED) {
            try {
                  gui.sendText("Requester: "+threadNumber+" is going
asleep for:"
                                    +processTime+" ms of simulated
traffic time.");
                  sleep(processTime);
                  gui.sendText("Requester: "+threadNumber+" is sending
SLSTableEntry.");
```

```
        //simulate server receiving FlowTermination message from router
                server.receiveSLSTableUpdate(response.getUserId());
            }
            catch (InterruptedException ie) {
                //do nothing
                gui.sendText(ie.toString());
            }
        }
        else {
            gui.sendText("Requester: "+threadNumber+
                        "'s request denied, result: "+result);
        }
    }

   /**
    * Returns the string representation of this class
    * @param none
    * @return string
    */
    public String toString(){
       return ("\nThread Number          : " + threadNumber +
               "\nProcess Time          : " + processTime +"");

    }//end toString()

}//end RequesterThread class

//end file RequesterThread.java
```

LIST OF REFERENCES

[1] Xie, Geoffrey G. and Lam, Simon S., "Delay Guarantee of Virtual Clock Server," IEEE/ACM Transactions on Networking, 3(6):683-689, December 1995.

[2] Xie, Geoffrey G., Hensgen, Debra, Kidd, Taylor, and Yarger, John, "SAAM: An Integrated Network Architecture for Integrated Services," paper presented at the 6th IEEE/IFIP International Workshop on Quality of Service, Napa, CA, [http://www.cs.nps.navy.mil/people/faculty/xie/pub]. May 1998.

[3] QingMing Ma and Peter Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees", Computer Science Department, Carnegie Melon University, Pitsburgh, PA 15213, USA.

[4] Xie, Geoffrey G., Hensgen, Debra, Kidd, Taylor, and Yarger, John, "Efficient Management of Integrated Services Using a Path Information Base." [http://www.cs.nps.navy.mil/people/faculty/xie/pub], 14 May 1998.

[5] Labovitz, C., Malan, G.R., and Jahanian, F., "Internet Routing Instability", Department of Electrical Enginerring and Computer Science, University of Michigan, Oct. 1998.

[6] Xipeng Xiao, Ni, L.M, "Internet QoS: A Big Picture", Michigan State University, Mar/Apr. 1999.

[7] Viswanathan, A., Feldman, N., Wang, Z., Callon, R, "Evolution of Multiprotocol Label Switching", May 1998.

[8] Adiseshu, H., Parulkar, G., Yavatkar, R., "A State Management Protocol for IntServ, DiffServ and Label Switching", Department of Computer Science, Washington University in St. Louis, 1998.

[9] Wei Zhao, Tripathi, S.K., "Routing guaranteed quality of service connections in integrated services packet networks", 1997.

[10] Apostolopoulos, G., Guerin, R., Kamat, S., "Implementation and performance measurements of QoS routing extensions to OSPF", 1999.

[11] Pornavalai, C., Chakraborty, G., Shiratori, N., "QoS based routing algorithm in integrated services packet networks", 1998.

[12] Zheng Wang, Crowcroft, J., "Quality-of-Service Routing For Supporting Multimedia Applications", Sept. 1996.

[13] Shigang Chen, Nahrstedt, K., "An overview of quality of service routing for next-generation high-speed networks: problems and solutions", University of Illinois, November/December 1998.

[14] Francols, "IETF Multiprotocol Label Switching (MPLS) Architecture", Cisco Systems, 1998.

[15] Dean Vrable and John Yarger, "The SAAM Architecture: Enabling Integrated Services", Computer Science Department, Naval Postgraduate School, Sept. 1999.

[16] Pornavalai, C., Chakraborty, G., Shiratori, N., "QoS Routing Algorithm for Pre-Computed Paths", 1997.

[17] Efraim Kati, "Fault Tolerant Approach for Development of Server Agent Based Active Network Management (SAAM)", Computer Science Department, Naval Postgraduate School, Mar. 2000.

[18] Raj Jain, "The Art of Computer System Performance Analysis", John Wiley & Sons, Inc., 1991.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Chairman,Code CS.....1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93940-5000

4. Dr. Geoffrey Xie.....2
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93943-5100

5. Dr. Bret Michael.....1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93943-5100

6. Mr. Cary Colwell.....1
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California 93943-5100

7. Head Librarian.....2
   DTT Library
   1 Depot Road, #02-01
   Defence Technology Tower A
   Singapore 109681

8. Henry C. Quek.....3
   Block 561, #13-271
   Pasir Ris Street 51
   Singapore 510561

9. Mustafa Altinkaya.....1
   Deniz Harp Okulu
   Yazilim Gelistirme Merkezi
   Tuzla Istanbul-TURKEY